SCIENTIFIC / FINANCIAL

SpicyRPN

User Manual

Multiple Precision (MP)

Scientific and financial calculation

Free Open Source Software

Original source and manual: MIT License

Bryan R. Davies

Copyright © 2026 Bryan R. Davies

Release 2.0.0

18 October 2026

Application version 2.0.0 (build 3)

Licensing and acknowledgments

Copyright © 2026 Bryan R. Davies, for this manual.

SpicyRPN and this manual. SpicyRPN Release 2.0.0 is Free Open Source Software. You may use, study, modify, and redistribute the original application and this manual under the **MIT License**, retaining its copyright and permission notice. The US$1.99 App Store price pays for the integrated, easy-to-install iOS distribution. The editable LaTeX files and figures are this manual's source form. Section E.1 contains the complete license. The source package's `LICENSING.md` describes its scope; third-party material retains its own terms.

GNU GMP 6.3.0. SpicyRPN uses the GNU Multiple Precision Arithmetic Library under its LGPL option. The GMP library is available under the GNU Lesser General Public License, version 3 or later, or the GNU General Public License, version 2 or later, at the recipient's choice. Its demonstration programs and test suite are licensed under the GNU General Public License, version 3 or later. See `https://gmplib.org/`.

GNU MPFR 4.2.2. SpicyRPN uses GNU MPFR for multiple-precision floating-point arithmetic. MPFR is licensed under the GNU Lesser General Public License, version 3 or later. Its documentation is licensed separately under the GNU Free Documentation License, version 1.3 or later, as specified in that documentation. See `https://www.mpfr.org/`.

GNU MPC 1.4.1. SpicyRPN uses the GNU Multiple Precision Complex library for complex arithmetic and elementary functions. MPC is licensed under the GNU Lesser General Public License, version 3 or later. See `https://www.multiprecision.org/`.

The mathematical libraries include work copyrighted by the Free Software Foundation and other contributors. MPC includes copyrights held by INRIA; MPFR credits the Pascaline and Caramba projects at INRIA. Their source and `AUTHORS` files contain detailed notices. Upstream documentation and support files retain their stated terms.

SQLite 3.51.2. SpicyRPN uses SQLite to store calculator state. The bundled SQLite amalgamation is in the public domain; its original notices remain with the source. See `https://www.sqlite.org/copyright.html`.

License texts. Chapter E contains the complete MIT, GPLv3, and LGPLv3 texts. Read the LGPL in section E.3 together with the GPL in section E.2, whose terms it incorporates. The libraries' source archives retain their individual license files and notices; see chapter D.

Warranty. SpicyRPN, this manual, GMP, MPFR, and MPC are provided **WITHOUT ANY WARRANTY**, including the implied warranties of **MERCHANTABILITY** or **FITNESS FOR A PARTICULAR PURPOSE**, to the extent permitted by applicable law. The applicable licenses contain the full warranty and liability terms.

Project and source information. The complete release source ZIP and compiled manual are available at no additional charge from `https://spicyrpn.com/`. LGPLv3 permits modifying the library portions and reverse engineering to debug those modifications. Section D.9 covers rebuilding and personal signing.

Contents

Licensing and acknowledgments ii

How to use this manual x

I Getting Started 1

1 Getting Acquainted with SpicyRPN 2
- 1.1 What the calculator provides . . . 2
- 1.2 Multiple precision in everyday use . . . 3
- 1.3 Installation and first launch . . . 3
- 1.4 A tour of the display . . . 4
- 1.5 Scientific and Financial keyboard panels . . . 6
- 1.6 Orange and blue shift functions . . . 8
- 1.7 Choosing RPN or ALG . . . 8
- 1.8 Flags and pending operations . . . 9
- 1.9 Help, keyboard input, and accessibility . . . 10
- 1.10 Your work is saved automatically . . . 12

2 The SpicyRPN Stack 13
- 2.1 Where the numbers wait . . . 13
- 2.2 Prepare a known starting state . . . 14
- 2.3 Entering a number and pressing ENTER . . . 15
- 2.4 Lift, drop, and the deepest register . . . 16
- 2.5 Functions and operand order . . . 16
- 2.6 Inspecting every register . . . 17
- 2.7 Swapping, rolling, and dropping . . . 18
- 2.8 Clearing X or clearing the stack . . . 19
- 2.9 Recovering operands with LAST X and LAST Y . . . 19
- 2.10 Choosing the stack depth . . . 20
- 2.11 Arriving from another RPN calculator . . . 21

3 First Calculations in RPN 22
- 3.1 Prepare a known starting state . . . 22
- 3.2 Enter numbers, signs, and exponents . . . 22
- 3.3 Add, subtract, multiply, and divide . . . 24
- 3.4 Squares, roots, reciprocals, and powers . . . 24
- 3.5 Continue from a result . . . 25
- 3.6 Keep two intermediate results on the stack . . . 25
- 3.7 Use the percentage keys . . . 26
- 3.8 Correct an entry or recover operands . . . 28

3.9 A short practice set 29
3.10 Where to go next 30

4 Calculations in ALG Mode 31
4.1 Entering and evaluating an expression 31
4.2 Precedence and parentheses 33
4.3 Powers and negative operands 34
4.4 Implicit multiplication 34
4.5 Functions before and after an operand 35
4.6 FUNC and multiple arguments 36
4.7 Editing an unfinished expression 37
4.8 ANS, continuation, and ordinary memory 39
4.9 Clearing and recovering from an error 40
4.10 Switching workspaces and choosing available functions 41
4.11 Financial assignments and explicit computation 42
4.12 Working with labeled worksheets 42

II Everyday and Scientific Use 45

5 Numbers, Precision, and the Display 46
5.1 Three meanings of precision 46
5.2 Setting working and display digits with PREC 47
5.3 Fixed, scientific, and engineering notation 48
5.4 Setting FIX in each entry mode 50
5.5 Reading long numbers 52
5.6 Display rounding and value rounding 53
5.7 Existing values and a new working precision 54
5.8 Choosing useful precision and recognizing limits 55

6 Storing and Reusing Values 57
6.1 The memory banks and their indicators 57
6.2 Ordinary storage and recall 58
6.3 Arithmetic storage 60
6.4 Reusing a constant 61
6.5 Complex values in memory 62
6.6 Unset registers and stored zero 62
6.7 Associative memory: values under numeric keys 63
6.8 Memories, working areas, and saved operands 65
6.9 The statistical and financial banks 66
6.10 Clearing the intended group with MCLR 67

7 Scientific Functions 69
7.1 Finding and applying a scientific function 69
7.2 Arithmetic, reciprocal, powers, and roots 71

7.3 Exponential and logarithmic functions . . . 72
7.4 Angle units and trigonometric functions . . . 73
7.5 Hyperbolic functions and their inverses . . . 75
7.6 Absolute value, integer parts, and rounding . . . 75
7.7 Two different remainder conventions . . . 77
7.8 Percentages and their retained base . . . 78
7.9 Factorials, permutations, and combinations . . . 79
7.10 Gamma and logarithmic factorials . . . 80
7.11 The error function and its complement . . . 81
7.12 Constants and random numbers . . . 81
7.13 Domains, errors, and useful checks . . . 82

8 Complex Numbers 84
8.1 Components, controls, and the display . . . 84
8.2 Entering and editing a complete pair . . . 85
8.3 Rectangular and polar forms . . . 88
8.4 Swapping components and changing signs . . . 90
8.5 Arithmetic and supported scientific functions . . . 90
8.6 Turning complex mode off . . . 92
8.7 Functions that use only real components . . . 93
8.8 Principal results and branch conventions . . . 94
8.9 Numerical limits and interpreting a result . . . 96

9 Unit Conversions 98
9.1 Choose the quantity and conversion direction . . . 98
9.2 What changes in RPN and ALG . . . 99
9.3 Length, area, volume, and mass . . . 101
9.4 US and Imperial names are part of the unit . . . 102
9.5 Temperature readings and temperature differences . . . 103
9.6 Motion, force, pressure, and torque . . . 104
9.7 Energy, power, time, and frequency . . . 105
9.8 Angle conversion and the angle setting . . . 106
9.9 Conversion factors, precision, and rounding . . . 107

III Statistical and Financial Calculations 109

10 Statistics 110
10.1 Prepare the statistical bank . . . 110
10.2 Enter observations in the right order . . . 111
10.3 What the six accumulators retain . . . 112
10.4 Correct or remove an observation . . . 113
10.5 Means and weighted means . . . 114
10.6 Sample standard deviations . . . 116
10.7 Fit a straight line . . . 116

10.8 Estimate a response and read correlation . . . 117
10.9 Inspect the totals and preserve useful results . . . 118
10.10 Statistics worksheets in ALG . . . 120
10.11 Stack effects, errors, and numerical limits . . . 121

11 Time Value of Money, Loans, and Savings 124
11.1 The five financial registers . . . 124
11.2 Cash signs, periods, and interest rates . . . 125
11.3 END and BEGIN payment timing . . . 126
11.4 When an RPN financial key stores and when it solves . . . 127
11.5 Clearing and preparing a financial problem . . . 128
11.6 Loan payments and a final balloon . . . 129
11.7 Present value and a payment budget . . . 130
11.8 Future value and regular savings . . . 131
11.9 Solving for the periodic interest rate . . . 131
11.10 Solving n and interpreting fractional periods . . . 132
11.11 Amortization: interest, principal, and balance . . . 134
11.12 TVM and amortization in ALG . . . 137
11.13 Money rounding, precision, and practical limits . . . 138

12 Cash Flows, NPV, and IRR 140
12.1 Amounts at equally spaced times . . . 140
12.2 Controls and a known starting state . . . 141
12.3 Entering individual and repeated flows . . . 142
12.4 Reviewing and replacing stored amounts . . . 144
12.5 Selecting the prefix with n . . . 145
12.6 Net present value . . . 146
12.7 Internal rate of return . . . 147
12.8 Bounds, multiple roots, and unsuccessful searches . . . 148
12.9 Cash-flow worksheets in ALG . . . 150
12.10 Comparing amount and rate results . . . 151
12.11 Clearing and numerical limits . . . 152

13 Additional Financial and Date Calculations 154
13.1 Controls and preparation . . . 154
13.2 Simple interest on 360- and 365-day bases . . . 155
13.3 Entering and reading calendar dates . . . 156
13.4 Adding or subtracting calendar days . . . 157
13.5 Actual and 30/360 day counts . . . 158
13.6 Bond inputs and coupon assumptions . . . 159
13.7 Calculating bond price and yield . . . 160
13.8 Preparing a depreciation calculation . . . 162
13.9 Straight line and sum of years' digits . . . 163
13.10 Declining balance and straight-line crossover . . . 164
13.11 Fractional useful lives and rounding . . . 166

13.12Financial and date worksheets in ALG 166
13.13Retained inputs, results, and error recovery 167

IV Reference and Problem Solving 169

14 Saved Work, Clearing, and Troubleshooting 170
14.1 What is saved automatically 170
14.2 Independent workspaces and shared data 171
14.3 Resuming an unfinished entry 173
14.4 Cancel the pending operation before editing 174
14.5 Choose the smallest clear that fits 175
14.6 Prepare a fresh problem without resetting the application 177
14.7 Diagnose an unexpected result 178
14.8 Recover from a calculation error 180
14.9 Respond to a saved-state error 182
14.10Identify the build and report a reproducible problem 184

15 Complete Command and Settings Reference 186
15.1 Reading an entry and following an example 186
15.2 Complete keyboard maps 188
15.3 Physical-keyboard shortcuts 191
15.4 Alphabetical reference: A through F 192
15.5 Alphabetical reference: G through N 201
15.6 Alphabetical reference: P through S 207
15.7 Alphabetical reference: T through Z and numeric legends 213
15.8 Display flags and pending indicators 215
15.9 Settings, defaults, and tool fields 216
15.10Complete conversion-unit inventory 218
15.11Domains and operational limits 220
15.12Errors, warnings, and recovery cues 222

A Learning and Mastering RPN 226
A.1 Infix, prefix, and postfix describe the same calculation 226
A.2 Prepare to see what each key does 227
A.3 Optional refresher: a waiting intermediate result 228
A.4 Translate grouping before translating keys 228
A.5 Choose an order that fits the stack 229
A.6 The project website's expression equal to 42 231
A.7 Use deepest-register repetition deliberately 233
A.8 Reverse subtraction or division safely 234
A.9 Give a reusable value an ordinary memory register 235
A.10 Practice, check, and recover 236
A.11 Answers and annotated checkpoints 236

B Why RPN? 241
B.1 A calculation built from intermediate results . . . 241
B.2 What ALG offers in return . . . 242
B.3 Counting keys fairly . . . 244
B.4 Learning effort and opportunities for error . . . 245
B.5 What the historical studies establish . . . 246
B.6 From Polish notation to pocket calculators . . . 247
B.7 Why SpicyRPN takes this form . . . 248
B.8 Sources and further reading . . . 249

C Mathematical and Numerical Reference 250
C.1 Finite multiple precision . . . 250
C.2 Rounding and integer conventions . . . 251
C.3 Real scientific definitions and domains . . . 252
C.4 Complex principal values . . . 254
C.5 Statistical accumulators and results . . . 258
C.6 Time value of money . . . 259
C.7 Amortization and simple interest . . . 262
C.8 Cash-flow discounting and internal rate of return . . . 262
C.9 Financial numerical root finding . . . 263
C.10 Gregorian dates and day counts . . . 264
C.11 Semiannual bonds . . . 265
C.12 Depreciation methods . . . 266
C.13 Conversion definitions and factor catalog . . . 268
C.14 Limits and independent checks . . . 270

D Obtaining and Building the Source 272
D.1 Preparing the development machine . . . 272
D.2 Obtaining the release source package . . . 274
D.3 Understanding the source layout . . . 275
D.4 Library deployment settings . . . 276
D.5 Building GMP, MPFR, and MPC . . . 276
D.6 Signing and personal runs . . . 279
D.7 Building and running on macOS . . . 280
D.8 Building for iPhone and iPad . . . 281
D.9 Rebuilding after modifying a library . . . 282
D.10 Running the native test suites . . . 283
D.11 Archiving and publishing the validated release . . . 286
D.12 Checking independent numerical references . . . 287
D.13 Building this manual . . . 288
D.14 Diagnosing a build problem . . . 289

E MIT and GNU license texts 291
E.1 MIT License . . . 291
E.2 GNU General Public License, version 3 . . . 292

E.3 GNU Lesser General Public License, version 3 304

Glossary and Mathematical Symbols **307**

Bibliography and Further Reading **332**

Index **337**

How to use this manual

SpicyRPN combines scientific and financial calculation with multiple-precision arithmetic. It offers two ways to work: reverse Polish notation (RPN), in which you enter operands before an operation, and algebraic entry (ALG), in which you write an expression. Choose the approach that suits the calculation and your experience.

You do not need to read this book from beginning to end. Start with the calculator's layout and the entry mode you intend to use, then turn to the functions you need. The main text explains everyday operation; the appendices provide extended RPN instruction, mathematical detail, and source-build information.

Choose your starting point

If you already use RPN, begin with the display and keyboard tour in chapter 1, then read chapter 2. Familiarity with another calculator is useful, but stack depth, entry, clearing, and operand recovery deserve particular attention in SpicyRPN. Afterward, go directly to the scientific, statistical, or financial functions you need.

If you are learning RPN, follow the orientation with chapters 2 and 3, in that order. The stack explains where your numbers go and which numbers an operation uses. Try the short calculations before moving to the more detailed instruction and practice in chapter A. For the reasoning and history behind this style of calculation, see chapter B.

If you prefer algebraic entry, read the orientation and chapter 4, then use the shared chapters on precision, scientific functions, complex numbers, and conversions. The memory, statistics, and financial chapters include ALG worksheet procedures as well as RPN examples. Follow the procedure for your entry mode: worksheet fields replace stack operand positions in ALG.

Keys, registers, and display notation

The manual distinguishes a key you press from a number or indicator you read. On a touch screen, "press" means tap the corresponding key. When a physical-keyboard shortcut is intended, the instructions identify it explicitly.

Notation	Meaning
ENTER	A calculator key or named operation.
f and g	The orange and blue shift keys.
X (S0 on the stack display)	The current RPN entry or result, shown as the main number.
Y (S1 on the stack display)	The next RPN stack register; inspect S1 in the full-stack view.
PV, PMT	Named financial registers.
5.0000	A display readout, with the shown formatting.
MEM	A flag or status indicator in the display.
RPN, ALG	The two entry modes.

X and S0 name the same position; so do Y and S1. The manual uses X and Y to match function names and operand formulas. LAST X and LAST Y recall saved operands from earlier operations; those saved values are separate from the values currently in S0 and S1. ALG worksheet fields and mathematical variables named X or Y are identified in their own context; they do not refer to the inactive RPN stack.

Read a key sequence from left to right. Consecutive key symbols mean successive presses, not keys held down together. When a sequence shows f or g, press that shift key first, then the key carrying the named orange or blue function. The function name, as well as the color, identifies the intended operation.

Numbers written as ordinary input, such as 125.75, mean enter those digits and the decimal point in order. A mathematical expression such as $3(4+5)$ describes a calculation; it is not a key sequence unless explicitly presented as one. In an RPN stack diagram, the register labels identify each value's position. Read those labels before tracing how an operation moves or combines values.

Following the examples

Check an example's starting conditions before entering its numbers. Entry mode, angle unit, precision, and display format can affect what you see. Financial examples also depend on cash-flow signs, the interest period, payment timing, and values already stored in the financial registers. Clearing the display alone does not clear every memory bank.

Follow an example's key sequence in order and compare the stated result with your display. A table of intermediate results lets you locate the first point at which your calculation differs. Where an example gives a rounded result, use the stated display format when comparing its final digits. A different display format can show the same underlying value in a different way.

For the distinction between working digits, displayed digits, and input accuracy, see chapter 5.

Finding a function or solving a problem

For an explanation and worked use, turn to the relevant subject chapter. For a key sequence, operands, mode availability, or side effects, use the command reference in chapter 15. The table of contents provides the book's structure; the index provides access by function and subject. In the PDF, the contents, cross-references, and bookmarks help you move between related topics.

Chapter 14 explains saved work, clearing operations, errors, and common causes of unexpected results. Chapter C gives the formulas and numerical conventions behind the calculations. Readers who want to build or modify SpicyRPN can turn to chapter D.

Application and manual versions

This manual is dated 18 October 2026 and identifies the application as version 2.0.0 (build 3). Match the application version and build when comparing behavior or reporting a problem. Include the entry mode, relevant settings, starting values, and key sequence so the calculation can be reproduced.

Both modes provide scientific, statistical, financial, conversion, and memory functions. RPN uses stack operands; ALG uses expressions, explicit computation, and labeled worksheets. The command reference identifies each mode's controls, and the individual chapters explain the required entry sequence.

Part I

Getting Started

Chapter 1

Getting Acquainted with SpicyRPN

SpicyRPN is a scientific and financial calculator for iPhone, iPad, and Mac. Its keyboard brings together everyday arithmetic, advanced mathematical functions, statistics, and financial calculations. You can work in reverse Polish notation (RPN) or enter a written expression in algebraic mode (ALG). Multiple-precision arithmetic lets you choose how many digits the calculator uses internally, while separate display settings control how many digits you see.

This chapter introduces the controls and the information on the display. There is no need to perform a calculation yet. If RPN is new to you, read the explanation of the stack in chapter 2 before trying the first calculations in chapter 3.

1.1 What the calculator provides

The same calculator can serve several kinds of work. Its scientific functions include powers and roots, logarithms, trigonometry, hyperbolic functions, factorials, and special functions such as Gamma and the error function. You can use real numbers or paired real and imaginary components, with rectangular or polar presentation for complex values. The conversion tool handles quantities such as length, mass, temperature, energy, and time.

For repeated work, ten ordinary memory registers hold values for later use. Associative memory identifies a stored value by the real component of a numeric key, in either entry mode. Separate banks hold statistical accumulators, financial inputs, and cash flows. These banks let you retain the data for one task while carrying out other calculations.

RPN is the initial entry mode and has a five-register stack by default. You can later choose a different fixed depth or an unlimited stack. ALG provides a separate expression editor with parentheses, precedence, functions, and a previous-answer value. Scientific functions, complex controls, ordinary memory, precision settings, and conversions are available in both entry modes. Statistics, finance, and associative memory use stack operands in RPN and explicit computation or labeled worksheets in ALG.

Table 1.1: Finding the main kinds of calculation.

Work	Where to begin
Everyday arithmetic	RPN stack and first calculations in chapters 2 and 3; algebraic entry in chapter 4.
Scientific and complex mathematics	Functions, angle units, and complex values in chapters 7 and 8.
Units and stored values	Conversions in chapter 9; memory operations in chapter 6.
Statistics	Samples, averages, standard deviations, and linear regression in chapter 10, in either entry mode.
Finance	Loans and savings in chapter 11; NPV and IRR in chapter 12; dates, bonds, and depreciation in chapter 13, in either entry mode.

1.2 Multiple precision in everyday use

Working precision is the number of digits available for the calculator's arithmetic. SpicyRPN starts with 600 decimal working digits and lets you select from 16 to 10 000.

Display precision is a presentation choice. The initial **FIX 4** setting normally shows four places after the decimal point, so the initial zero appears as `0.0000`. A short display does not mean that the calculation used only four decimal places. For example, a value near 1/3 can appear as `0.3333` while many more digits are retained internally. Fixed, scientific, and engineering notation offer different ways to present the same stored value.

The **PREC** tool separates Working digits from Display digits. Open it with **g** followed by **7**, the key carrying the blue **PREC** legend. You can start with the defaults; chapter 5 explains when and how to change them.

The word INFIX

The help text uses **INFIX** to refer to working precision. In this context it does not mean that ALG is selected. Read the RPN/ALG buttons above the display to identify the active entry mode.

1.3 Installation and first launch

SpicyRPN Release 2.0.0 supports iOS/iPadOS 18.0 and later, and macOS 15.0 Sequoia and later on compatible Apple-silicon and Intel Macs. The Mac application includes both arm64 and x86_64 code. The US$1.99 App Store price pays for

the integrated, easy-to-install iOS distribution. Install it using the App Store on a supported device. Release 2 follows the original SpicyRPN application, which appeared on the App Store around 2019.

The matching source ZIP and compiled manual are available at `https://spicyrpn.com/`. You can build the application with Xcode on a Mac supported by the selected Xcode version. The source-build procedure prepares the included mathematical libraries for the intended destination, then builds the SpicyRPN scheme for Mac, an iPhone or iPad simulator, or a physical iPhone or iPad. A physical-device build requires signing configuration. Chapter D gives the complete instructions.

When you open an installation with no saved calculator state, the calculator shows zero and the Scientific panel. Its initial settings are listed in table 1.2. On later launches, SpicyRPN restores your saved work instead of returning to these defaults.

Table 1.2: The starting settings for a new calculator state.

Setting	Initial choice
Entry mode	RPN
Stack	Five registers, initially zero
Numbers	Normal real-number mode
Angle unit	Degrees, shown as **DEG**
Display	**FIX 4**, initially `0.0000`
Working precision	600 decimal digits
Date order	Month/day/year, shown as **MDY**
Payment timing	End of period; **BGN** is absent

Identifying your version

This manual covers application version 2.0.0, build 3. Open **HELP**, choose Licenses & Acknowledgments, then Credits to read the installed version and build.

Keep both numbers when reporting a problem. If you built the application yourself, also record the source package version and any local changes. Include the entry mode, settings, starting values, and key sequence with a report; the version alone does not describe a calculation.

1.4 A tour of the display

The display is an instrument panel: the large number is only one part of the information it provides. Read the entry mode above it, the flags along its top edge, and the small detail and status lines below the number. Figure 1.1 shows the arrangement in a normal real-number RPN view.

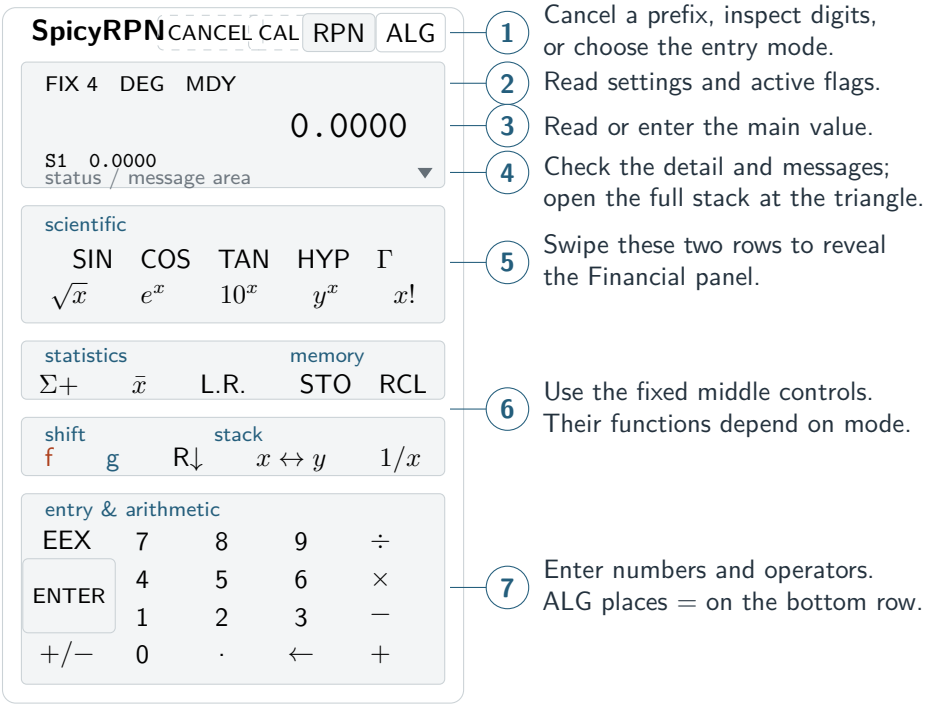


Figure 1.1: Schematic of the portrait RPN layout, with the Scientific panel selected. The numbered notes identify display regions and key groups; alternate key legends are omitted. Dashed header controls appear only when needed: CANCEL for a pending selection and CAL for an overflowing number. This is a layout guide, not a screen capture.

The header contains the RPN and ALG buttons; the selected button is emphasized. While a shift, multi-key command, or ALG worksheet is active, **CANCEL** appears beside the branding. It cancels a pending command without changing your calculation, or closes a worksheet while preserving its accepted work. When a number is too long to fit, the header also offers **CAL**, which shows or hides a digit ruler.

The flag row identifies settings such as **FIX 4**, **DEG**, and **MDY**. Additional flags appear for active shifts, unfinished entry, pending commands, and occupied memory banks. They are explained in section 1.8.

The large number is the active entry or result. In RPN it is X, listed as S0 in the full-stack display. Open the full-stack view to read Y at S1 and to inspect deeper registers. Real RPN devotes the main display to larger X digits; complex RPN shows both components of X. Thus an instruction to read X means read the main value or S0; an instruction to read Y means read S1. These are two naming conventions for the same positions, not separate registers. Chapter 2 explains how values move between them. In ALG, the smaller line shows the expression instead, in both real and complex mode.

The status area supplies prompts and messages. A command such as **STO** uses it to tell you what the calculator expects next. An error or a complex-number warning can occupy this area, so read it when the result or the next key's behavior is unexpected.

The small triangle at the lower right opens the full stack in RPN. Its rows are labeled S0, S1, S2, and so on: X is S0, Y is S1, and larger numbers are deeper positions. Opening this view does not perform a calculation. Close it with its collapse control or by pressing outside the expanded panel. ALG has no RPN stack inspector.

Long numbers scroll horizontally. Drag the digits to inspect more of the number; an exponent remains pinned in place. In real RPN, the **CAL** ruler appears only for an overflowing number and shares the bottom row with the stack triangle. The ruler marks groups of five digits. Complex mode provides two component rows for the current value. Both ALG displays place the expression on the bottom line, below calibration, and use digit dragging without scrollbars. The expression is also available to VoiceOver. Complex RPN retains its scrollbars. See chapters 5 and 8 for these displays in detail.

1.5 Scientific and Financial keyboard panels

In portrait, the top two keyboard rows form a sliding panel. Swipe *left across those rows* to reveal Financial; swipe right to return to Scientific. The remaining controls stay in place. The Scientific panel contains functions such as **SIN**, **COS**, roots, powers, and logarithms. The Financial panel contains the five time-value-of-money

keys and the cash-flow keys, with further financial and date operations on their shifted legends.

In a sufficiently wide landscape view, both panels are visible in the left bank. In RPN, the right bank contains the shift and stack controls, followed by entry and arithmetic. In ALG, the Expression row sits below the two panels on the left; Shift and Memory sit above entry and arithmetic on the right. Resizing a Mac window can change the arrangement in the same way. Across the two panels and fixed controls, RPN has 49 key positions and ALG has 50; portrait shows 39 in RPN or 40 in ALG. The keyboard itself does not scroll vertically.

On iPad, the calculator fills the available area with proportionally spaced keys, larger labels, and an enlarged display. The layout adapts when you rotate the device, retaining the same functions and panel behavior described here.

Changing panels only changes which keys are visible. It preserves an unfinished number or expression and does not consume an active shift or a pending command. If **STO** is showing when you swipe, for example, it will still be waiting for a storage destination afterward.

A panel is not an entry mode

Scientific and Financial are keyboard panels. RPN and ALG are entry modes, selected above the display. Showing the Financial panel does not select RPN. In ALG, financial registers use explicit storage, RCL, and CPT; multi-input operations open worksheets. The shared formulas retain their stated period and date conventions.

Key-click feedback

Enabled calculator keys play a soft click in both RPN and ALG, whether you tap a key or use its physical-keyboard shortcut. This includes shifts, pending-command selections, and operations that report an error. The header's RPN, ALG, **CAL**, and **CANCEL** buttons also click, as does Escape when canceling or clearing calculator input. Disabled keys, unhandled keyboard characters, scrolling, and changing panels do not play a click.

The click confirms a key press; it does not indicate that a calculation succeeded or change any stored value. Its audibility depends on the system's audio settings; on iPhone and iPad, clicks respect Silent mode and mix with background audio. The **GOAT** command plays its separate sound independently of key clicks; see GOAT on page 201.

1.6 Orange and blue shift functions

Many keys carry a main legend and one or two alternate legends. Press the key by itself for its main function. Press the orange f first for its orange function, or the blue g first for its blue function. These are successive presses: you do not hold the shift key while pressing the next key.

The 7 key illustrates the arrangement:

Press	Selected function
7	Enter the digit 7.
f 7	Select FIX, the orange legend. Its entry procedure depends on the entry mode; see chapter 5.
g 7	Open PREC, the blue legend, to inspect or change working and display digits.

The display's **f** or **g** indicator confirms the selected shift. Pressing the same shift again cancels it; pressing the other shift selects that other layer. A shift normally applies to one following key, then releases. The chosen function may itself begin a command that needs another key, such as MCLR or inverse hyperbolic selection. If a key has no legend for the selected layer, the calculator reports that it has no such alternate function.

To become familiar with shifting without changing a numerical value, start with no pending command, no active shift, and no tool open. Press g 7. The Precision & display panel opens. Read its two fields, then press Close without changing them. This is also a convenient way to check the current working precision.

1.7 Choosing RPN or ALG

An ordinary mode selection cancels active shifts and pending command prefixes. Finish or cancel a ΔFIX digit-count prompt before switching: while that prompt is active, the first mode-button press cancels the prompt and leaves the entry mode unchanged.

Select RPN or ALG with the buttons in the header. In RPN, you enter the operands before choosing an operation. The stack holds those operands and intermediate results. In ALG, you build an expression and press = to evaluate it. In ALG, the left entry column has normal-size EEX, $1/x$, +/−, and = keys from top to bottom, placing = on the bottom row. Orange $1/x$ selects INT; blue $1/x$ selects FRAC. = has no shifted legends.

Other controls change with the mode. ALG has one five-key Expression row: (,) , FUNC , CPT , and CLEAR . In portrait it sits above f , g , and the three Memory keys ANS , STO , and RCL , in that order. The Scientific panel, number keys, arithmetic keys, and ordinary memories remain available. The CPT main key starts an explicit financial computation; orange STAT and associative-memory legends open labeled worksheets. See sections 4.11 and 4.12.

The two modes preserve independent working areas. Switching modes does not evaluate an unfinished expression, finish an unfinished number, or copy the displayed operand into the other mode. When you return, that mode's work is still available. Precision, angle and display settings, complex mode, and memory banks are shared, however. A change to a shared setting in one mode also applies in the other.

The statistical and financial chapters give RPN examples and separate ALG worksheet procedures. If you prefer written expressions, use ALG and continue with chapter 4. For RPN, continue with chapter 2; it explains where the numbers go before you begin calculating.

1.8 Flags and pending operations

A flag can describe a setting, announce stored data, or explain why the calculator is waiting for another key. These meanings are different. For example, **MEM** means that an ordinary memory register contains stored data; **STO** means that a storage command is waiting for its destination. Neither means that you should type a new number without first checking the pending command.

Table 1.3: The main display indicators.

Indicator	What it tells you
FIX, **SCI**, **ENG** and a count	The notation and display-digit setting.
DEG, **RAD**, **GRAD**	The selected angle unit: degrees, radians, or gradians.
MDY, **DMY**	Month/day/year or day/month/year date order.
COMPLEX	Complex-number mode is active.
BGN	Payments occur at the beginning of each period. Its absence means end-of-period timing.
f, **g**	An orange or blue function is selected for the next key.
ENTRY	A number is being entered in RPN, or an ALG expression is unfinished.
STO, **RCL**	Choose a register to store in or recall. A modifier in **STO+**, **STO−**, **STO×**, or **STO÷** identifies arithmetic storage.
HYP, **INVHYP**	Choose SIN , COS , or TAN for a hyperbolic or inverse hyperbolic operation.

Indicator	What it tells you
FUNC	ALG is waiting for the function to insert before its arguments.
CPT	ALG is waiting for a TVM register, NPV, or IRR to compute.
ΔFIX	ALG is waiting for a display-digit count, accepted with =.
MCLR	The next key selects a group to clear.
MEM, **ASSOC**	Ordinary registers or associative memories contain stored entries.
STAT, **FIN**, **CF**	Statistical registers, financial registers, or cash flows contain stored entries.

Memory flags indicate occupied storage, including an explicitly stored zero. They do not report whether the stored values are suitable for the calculation you are about to perform. The flags also remain meaningful in ALG: a **STAT** or **FIN** flag shows shared bank contents, including data entered in either mode.

Recognizing and canceling a command

Read the status prompt whenever a command flag appears. Press the header's CANCEL button to dismiss any pending shift or command, including **STO**, **RCL**, **MCLR**, **HYP**, **FUNC**, **CPT**, and the ALG FIX digit-count prompt. A shift and command selected together are both canceled. Your unfinished number, expression, stack, memories, and settings remain unchanged; a displayed error is also dismissed. The cancellation is saved for relaunch.

CANCEL appears while one of these selections is waiting for a key or an ALG worksheet is active. The worksheet's own CANCEL and the header's CANCEL both discard unaccepted field edits and return to the preserved main expression; previously accepted values and completed computations remain. Typing a number or leaving an ALG expression unfinished does not, by itself, make CANCEL appear.

Use CANCEL when you want to leave a pending selection without editing your numbers. Backspace, shifts, and Escape have additional context-dependent uses; section 14.4 gives the complete cancellation guide. Clearing memory groups is covered in chapter 6.

1.9 Help, keyboard input, and accessibility

For a quick reminder, start with no pending command or active shift and press g RCL, selecting its blue HELP legend. HELP opens the RPN key guide or

ALG key guide for the current mode. Each guide covers that mode's primary and shifted keyboard functions, grouping related functions together. Keys from the other mode are not included. Both guides have three topics: Entry, Memory, and Tools. Each topic has separate, spaced entries with key-style labels: black for primary functions, orange for orange-shifted functions, and blue for blue-shifted functions. Scroll within the topic to read further entries; the licensing link remains fixed below the scrolling content. Select a topic, then use **Close** to return to the calculator. Parameter panels such as **PREC** also provide guidance above their fields. They show an **Apply** button to accept changes and keep the panel open if an input needs correction.

The Licenses & Acknowledgments link at the bottom of HELP opens a separate sheet with Credits, MIT, GPLv3, and LGPLv3 tabs. Each tab scrolls; the full license texts are available offline. Select **Done** to return to HELP without changing your calculation. MIT covers original SpicyRPN material; the GNU licenses cover the libraries. The same complete license texts appear in chapter E.

Using a physical keyboard

A Mac keyboard or a connected keyboard can enter numbers and many commands. Click or tap the display to give calculator input focus if typing is not reaching it. When the stack inspector is open, physical Escape closes it; calculator input is held back while you inspect the stack. Inside a parameter panel, keyboard input belongs to that panel. The shortcuts select the corresponding calculator keys, so an active **f** or **g** also affects a following digit or operator typed on the physical keyboard.

Table 1.4: Frequently used physical-keyboard input.

Physical key	Calculator action
`0`–`9`, `.`	Digits and decimal point.
`+`, `-`, `*`, `/`	Addition, subtraction, multiplication, and division.
`e` or `E`	**EEX**, for exponent entry.
`s` or `S`	+/−, to change sign.
`f`, `g`	Orange and blue shift.
Return or `=`	**ENTER** in RPN; = in ALG.
Backspace/Delete	The calculator's backspace key.
`(`, `)`, `,`	Parentheses and a function-argument separator in ALG.
Escape	Cancel an active shift, pending command, or error message; with none active, clear X or the current entry.

The letter `e` enters an exponent, not Euler's constant, and the minus key selects subtraction rather than the sign-change key. Use `s` for sign changes, including an exponent's sign. The complete shortcut list is in chapter 15.

Accessible controls

Keys provide spoken function names, including names for mathematical legends. Their accessible names follow the selected orange or blue layer, and shift keys report whether they are selected. The mode buttons identify the active entry mode. Display values, exponents, status messages, and errors also have accessible labels and values.

The portrait panel offers named actions Show scientific panel and Show financial panel, so changing panels does not depend on a drag gesture. The display offers Focus numeric entry; the RPN triangle is named Show full stack. Number and stack scroll controls provide adjustable actions for moving through content. The interface also respects the system's Reduce Motion setting for panel and key animations. Use these names when navigating with assistive technology rather than relying on color or position alone.

1.10 Your work is saved automatically

SpicyRPN saves calculator actions as you work, including individual keystrokes, in a local database belonging to the application. There is no separate Save step before closing it. Saved work includes the RPN stack, the independent ALG expression and answer, memories, settings, unfinished entry, and pending commands. Active worksheets also resume with their saved fields and results.

Reopening the application therefore resumes a working calculator, which may still be waiting for the next digit or the destination of a **STO** command. Read the mode, flags, and status prompt before continuing. Closing the application is not a clearing operation.

If a save fails, the calculator reports the problem and attempts to restore the last successfully saved state. If it cannot restore the database, it reports an error instead of silently starting a new calculator. Follow the saved-state guidance in chapter 14 when such a message appears.

With the layout familiar, the next step for RPN is to understand the stack in chapter 2. ALG users can continue directly to chapter 4.

Chapter 2

The SpicyRPN Stack

The stack holds the numbers you are working with: operands awaiting a function and results awaiting their next use. In RPN, you enter the operands first, then press the function that uses them. There is no equals key to finish an expression. Each function acts immediately on the values in the required stack positions.

This chapter explains those positions and the movements between them. The short examples make the movements visible; chapter 3 then puts them to work in complete calculations. These rules describe the RPN workspace. ALG has its own entry engine and retains its own work when you switch modes.

2.1 Where the numbers wait

The main displayed number occupies S0, conventionally called X. The next position is S1, conventionally called Y. This manual uses both names: **X means S0, and Y means S1**. A single-operand function uses X; a two-operand function uses Y and X. Larger displayed register numbers identify successively deeper positions. These are names for positions, not permanent names for particular values. As the stack moves, a value can move from X to Y or back again.

SpicyRPN starts with a fixed stack of five registers, all containing zero. Figure 2.1 shows their order. Throughout this manual, stack diagrams put X/S0 at the bottom, matching the expanded stack display. The word *deepest* means the register farthest from X: S4 in this five-register stack. It appears at the top of the diagram.

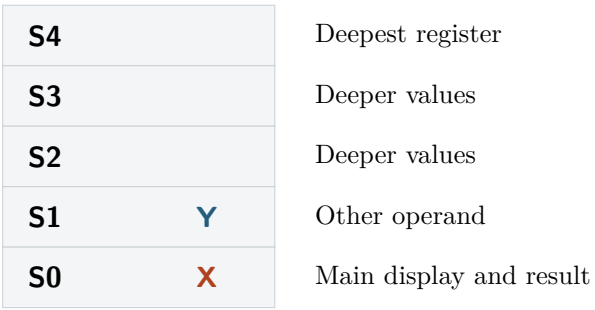


Figure 2.1: The default five-register stack. The displayed names count outward from S0; X stays at the bottom of every stack diagram in this chapter.

Each position holds one complete number. A complex number also occupies one position: its real and imaginary components travel together, as do its magnitude

and angle when shown in polar form. Changing the number of displayed decimal places does not change the value held in a register. See chapters 5 and 8 for those details.

The stack is working space, distinct from ordinary, statistical, and financial memory registers. A value that must remain available while you rearrange or clear the stack belongs in memory; see chapter 6.

2.2 Prepare a known starting state

SpicyRPN restores saved work when it opens. Before trying the examples, prepare the calculator using the steps below; a new launch does not clear it. The examples in this chapter, chapter 3, and chapter A use RPN, real-number mode, a fixed five-register stack, 600 working digits, and FIX 4. Angle units, date order, and payment timing do not affect these examples.

The fields in **STACK** and **PREC** are typed settings, not numbers entered on the calculator's stack.

1. Cancel any open worksheet, then close any tool or stack inspector. Select RPN above the display. This cancels ordinary pending commands and shifts. If an ALG **ΔFIX** prompt was active, the first press cancels that prompt; press RPN again and check the selected mode. There should be no **f**, **g**, or pending-command flag.
2. Press **f** **CLEAR**, the orange function of $1/x$, to clear the stack and any unfinished RPN number. If **COMPLEX** is showing, press **f** **CMPLX**, the orange function of $\div$, to return to real mode. If this reports unfinished ALG input, return to ALG, finish, correct, or clear that entry, then repeat the setup from step 1.
3. Press **f** **STACK**, the orange function of $x \leftrightarrow y$. Set Depth (0 = unlimited) to 5 and press **Apply**.
4. Press **g** **PREC**, the blue function of **7**. Set Working digits to 600 and Display digits to 4, then press **Apply**. To select fixed notation as well, enter 4 **f** **FIX**; **FIX** is the orange function of **7**. This uses and removes the digit-count operand.
5. Show the Scientific panel if necessary, then press **f** **CLEAR** once more. The main display should read `0.0000`, with **FIX 4** and no **ENTRY** or pending-command flag.

In these introductory examples, *the prepared start* means these settings, no active shift or pending command, and a stack cleared with **f** **CLEAR**. Set up once, then repeat CLEAR before each independent example; do not clear between successive steps of one calculation. CLEAR preserves settings, memories, and saved operands,

so memory flags may remain lit. The examples establish the operands they need. Section 2.8 and chapter 6 explain clearing scope.

2.3 Entering a number and pressing ENTER

Starting a new number reserves a new X position. On the first digit, decimal point, or **EEX**, the previous X moves to Y, the previous Y moves to S2, and so on. This movement is called *stack lift*. Further characters edit that same number; they do not lift the stack again. While you are typing, **ENTRY** is shown and the main display shows the entry as typed.

ENTER has two distinct uses:

- **During number entry,** it finishes the number in X. The stack has already lifted, so this press does not lift again or duplicate the number.
- **When no number is being entered,** it duplicates X by lifting a copy onto the stack. The original value becomes Y.

To enter two different operands, type the first number, press **ENTER**, then type the second. A function key finishes an active entry before calculating, so you can press the operator immediately after the second number. Typing more digits without **ENTER** would extend the first number instead.

Watch two operands take their places

Use the prepared start in section 2.2, then enter:

2 **ENTER** 3 **+**

The table follows each press. Integers stand for the stored values; the final display is `5.0000`. Italics identify a number still being entered.

Position	Cleared	**2**	**ENTER**	**3**	**+**
S4	0	0	0	0	0
S3	0	0	0	0	0
S2	0	0	0	0	0
S1/Y	0	0	0	2	0
S0/X	0	*2*	2	*3*	5

To duplicate a freshly typed number, use **ENTER** twice: the first press confirms it, and the second duplicates it. For example, starting with the same settings and a cleared stack, 6 **ENTER** **ENTER** **+** gives `12.0000`. After a completed result, one **ENTER** is enough to duplicate it.

2.4 Lift, drop, and the deepest register

A *lift* makes room at X by moving existing values one position deeper. A *drop* moves deeper values toward X. An ordinary two-operand operation combines Y and X into one result in X; the old S2 becomes Y, and the remaining deeper values follow it down.

A fixed stack keeps exactly its configured number of positions. A lift therefore discards the previous deepest value. A drop instead repeats the deepest value to fill the vacated position. With five registers, the old S4 moves to S3 and remains in S4 as well. A drop does not fill the deepest position with zero unless the value already there was zero.

See both ends of a fixed stack

Use the same RPN, real-mode, fixed-depth-5, FIX 4 settings, with no pending command. Prepare the starting column of table 2.1 with:

f CLEAR 5 ENTER 4 ENTER 3 ENTER 2 ENTER 1 ENTER

The final ENTER confirms 1. Each action column below is a separate experiment from that starting column; repeat the preparation before trying another column.

Position	Start	ENTER	9 ENTER	+
S4	5	4	4	5
S3	4	3	3	5
S2	3	2	2	4
S1/Y	2	1	1	3
S0/X	1	1	9	3

Table 2.1: Independent actions on a full five-register stack. Both lift examples discard 5; addition repeats the deepest 5.

2.5 Functions and operand order

A *single-operand* (unary) function uses X and replaces it with the result. The rest of the stack stays in place. Square root, reciprocal, and sine are examples. A *two-operand* (binary) arithmetic function uses both Y and X, leaves its result in X, and drops the deeper values by one position.

For subtraction and division, the order is $Y - X$ and Y/X. For a power, Y is the base and X is the exponent. For an indexed root, Y is the radicand and X is

the root index. Enter the left-hand operand first. Table 2.2 gives complete short sequences.

Table 2.2: Operand order in real RPN. For each row, use a five-register stack and FIX 4, with no pending command, and first press f CLEAR. Results are shown without the display's four trailing decimal places.

Calculation	Keys after clearing	Result
$9 - 4$	9 ENTER 4 $-$	5
$12/3$	12 ENTER 3 $\div$	4
2^3	2 ENTER 3 y^x	8
$\sqrt[3]{8}$	8 ENTER 3 f $\sqrt[x]{y}$	2
$\sqrt{9}$	9 $\sqrt{x}$	3

If the operands are reversed, exchange them with SWAP (the $x \leftrightarrow y$ key) before pressing the operator. The distinction matters for subtraction, division, powers, and roots even when swapping would leave a sum or product unchanged.

Not every function consumes its operands

The percentage functions %, $\Delta\%$, and T% use X and Y but replace only X; Y and all deeper positions remain in place. For example, under the settings above, f CLEAR 200 ENTER 10 g % leaves 20 in X and 200 in Y. Pressing + next gives 220.0000.
Other specialized commands have their own stack effects; the reference note near the end of this chapter points to their explanations.

2.6 Inspecting every register

The main display shows X. To inspect S1/Y and all other positions, press the small triangle at the lower-right corner of the display. Its accessibility label is "Show full stack." The expanded display opens at S0, with deeper registers above it, in the order shown in figure 2.1.

Scroll vertically to reach deeper positions. Each long number can be scrolled horizontally in its own row, with its exponent remaining visible. Real-number rows use larger digits with a ruler near the bottom when the number overflows; drag the digits to scroll without a scrollbar. In complex mode, each register shows both components of its one value. VoiceOver identifies the register with each component, for example, "Stack register S0, real component" and "Stack register S0, imaginary component." In polar form, those labels identify the register's magnitude and angle instead. Close the view with its triangle, labeled "Collapse stack," or by tapping outside the expanded panel.

Inspection does not calculate, move values, finish an entry, or consume a pending command. If you open it midway through typing, S0 shows the unfinished entry and S1 already contains the preceding X. Use this view whenever you want to check operand positions without rolling the stack.

2.7 Swapping, rolling, and dropping

The stack controls let you arrange existing values without doing arithmetic. Their locations are collected in table 2.3. When a number is being entered, these commands finish it before acting on the stack.

Roll is available only with a fixed stack. **SWAP** works with an unlimited stack when at least two positions exist.

Operation	Key sequence	Effect
SWAP	$x \leftrightarrow y$	Exchange X and Y; deeper values stay in place.
Roll down	**R↓**	Move Y to X and each deeper value toward X; wrap old X to the deepest position.
Roll up	**g** **R↑**	Move each value one position deeper; wrap the deepest value to X.
DROP	**f** **DROP**	Remove X; move Y and deeper values down.

Table 2.3: Stack controls. **DROP** and roll up are the orange and blue functions of the roll-down key.

Both roll directions are cyclic: no value is lost or duplicated. Rolling once each way restores the original order. Rolling in one direction five times restores a five-register stack; for another fixed depth, use that many rolls.

DROP discards X instead of wrapping it around. In a fixed stack it repeats the deepest value, as described in section 2.4. In an unlimited stack it shortens the stack by one; dropping its last value leaves a single zero in X.

Table 2.4 uses the same prepared stack and settings as table 2.1. Repeat that preparation before each action; the columns are independent results, not successive steps.

Position	Start	SWAP	Roll down	Roll up	DROP
S4	5	5	1	4	5
S3	4	4	5	3	5
S2	3	3	4	2	4
S1/Y	2	1	3	1	3
S0/X	1	2	2	5	2

Table 2.4: Rearranging a five-register stack with one command.

2.8 Clearing X or clearing the stack

Check a pending command before clearing

If STO or RCL is waiting for a target, backspace, CLx, or CLEAR cancels that pending command without clearing a value. Repeat the desired clearing sequence to clear. CANCEL in the header dismisses any pending shift or command without clearing a value. Backspace also cancels MCLR; its other selectors clear groups as described in chapters 6 and 14.

CLx, the blue function of the backspace key, replaces X with zero. It leaves every deeper position in place and saves the previous X in Last X. If a number is still being entered, CLx first finishes that number, then saves and clears it. Last Y is unchanged.

CLEAR, the orange function of $1/x$, clears the whole RPN stack and discards any unfinished entry. A fixed stack retains its configured depth, filled with zeros; an unlimited stack returns to one zero in X. CLEAR preserves both Last X and Last Y. It does not clear ordinary, associative, statistical, or financial memory, and does not reset settings.

Neither operation removes an X position in the way DROP does. After CLx, starting a new number lifts the zero just created into Y. Use backspace to correct individual digits while an entry is active. Deleting the last character ends a real-number entry with zero in its reserved X position; it does not reverse the lift that began the entry. When no entry is active, backspace acts as CLx.

2.9 Recovering operands with LAST X and LAST Y

The registers Last X and Last Y hold saved operands outside the working stack. They are not the current S0/X and S1/Y positions. Recall LAST X with the blue function of $x \leftrightarrow y$; recall LAST Y with the blue function of $1/x$. Each recall lifts the saved value into X, moving the current result to Y. Recalling either saved operand leaves both saved values unchanged.

Successful single-operand arithmetic saves the original X and also the current Y, if that position exists. Two-operand arithmetic, including percentages, saves both operands. A later calculation can therefore replace the saved pair. With an unlimited stack containing only X, a single-operand function updates Last X and leaves Last Y unchanged.

Entry, ENTER, SWAP, roll, DROP, and CLEAR preserve the saved operands. Constants and random-number generation preserve them too. CLx and ordinary memory recall replace Last X with the previous X, leaving Last Y alone. Specialized commands are covered in the reference note below.

Recover the operands of a mistaken addition

Use RPN, real mode, fixed depth 5, and FIX 4, with no pending command. Start with:

f CLEAR 12 ENTER 3 +

The result is 15.0000. The saved operands are Last X = 3 and Last Y = 12. To calculate the division you intended, remove the mistaken result, recall the first operand, recall the second, and divide:

f DROP g LAST Y g LAST X ÷

Before division, Y contains 12 and X contains 3. The result is 4.0000. DROP and the two recalls preserve the saved pair until division uses it.

These are operand-recovery commands, not general undo. They do not restore a complete earlier stack, recover values lost beyond a fixed stack's deepest position, or reverse a memory or settings change. Recover the operands before another operation overwrites them.

2.10 Choosing the stack depth

Changing to a larger fixed depth adds zero-filled positions at the deep end. Changing to a smaller fixed depth keeps X, Y, and as many following positions as fit; deeper values are discarded. Changing from a fixed stack to unlimited keeps every existing position, including zeros. It does not remove padding left by the fixed stack. Changing from unlimited to fixed similarly retains the positions nearest X, adding zeros or discarding deeper values to reach the chosen depth.

To change the depth, press f STACK, the orange function of $x \leftrightarrow y$. In the field labeled "Depth (0 = unlimited)," enter a whole number from 2 through 1000 for a fixed stack, or 0 for an unlimited stack, then press Apply. The number in this field is a setting, not an operand you enter on the calculator's stack. Applying the setting finishes any active number entry before changing the depth.

An unlimited stack grows on each lift and shrinks on each ordinary binary operation or **DROP**. There is no deepest-register repetition and no fixed capacity at which a lift discards a value. When it would otherwise become empty, **DROP** leaves one zero in X. Starting a new number still lifts that zero; zero is an ordinary stack value. A two-operand function or **SWAP** requires at least two positions, and reports insufficient operands if only X exists. Roll up and roll down are unavailable in unlimited mode, regardless of its current number of positions.

Choose a fixed depth when you want a bounded stack with cyclic roll and deepest-register repetition. Choose unlimited when you want pending values to remain until explicitly consumed or dropped. Neither choice replaces memory for values you want to name and reuse across calculations.

Reference: complex values and other commands

The introductory examples use real numbers. If a stack value retains an imaginary component from earlier complex work, moving it preserves that component. With CMPLX off, however, an idle ENTER creates a real-only copy, and Last-register captures and recalls also write only the real component. Toggling CMPLX alone preserves existing saved operands. Section 8.6 explains these distinctions with a worked example.
Ordinary memory recall, LAST X, LAST Y, constants, and random-number generation lift a new value into S0/X. Associative recall instead replaces the key there; see chapter 6. A TVM solve uses financial registers and lifts its answer; both TVM solves and financial-register recalls preserve the saved Last operands. Statistics and other financial commands can replace one value or return several. Their chapters specify the effects: chapters 10 to 13.

2.11 Arriving from another RPN calculator

If you are accustomed to a traditional four-register HP calculator, the familiar X/Y operand order and fixed-stack repetition will help you get started. SpicyRPN adds a fifth register by default, displays positions as S0, S1, and so on, and lets you choose a different depth. You can select four registers if that capacity suits your work.

Pay particular attention to the timing of entry: the first digit lifts, the first **ENTER** confirms the entry, and a further **ENTER** duplicates it. Also remember that **CLx** saves the cleared X, while **CLEAR** preserves both saved operands. Use the full-stack view to check these effects as you develop a feel for the calculator. Chapter 3 begins the practical calculations; chapter A provides extended practice, and chapter B discusses the reasons and history behind RPN.

Chapter 3

First Calculations in RPN

The stack rules become useful as soon as you have a calculation to do. This chapter takes you from entering one number to combining intermediate results, applying percentages, and repairing a mistake. The main value, S0 (called X), and the next register, S1 (called Y), are the two stack positions used by the next ordinary arithmetic operation. Open the full-stack view whenever you want to check both operands, then close it to continue.

The examples build on chapter 2. You do not need to plan every stack movement in advance: enter the numbers needed for one step, perform that step, and use its result in the next one. More extensive instruction in translating written expressions appears in chapter A.

3.1 Prepare a known starting state

Use the preparation procedure in section 2.2: RPN, real mode, a fixed five-register stack, 600 working digits, and FIX 4. If you have just completed the stack chapter, keep those settings. SpicyRPN restores saved work at launch, so check them if you are returning later. Angle units, date order, and payment timing do not affect this chapter's answers.

Throughout this chapter, *the prepared start* means those settings, no active shift or pending command, and a stack cleared with f CLEAR. Repeat CLEAR before each independent example, but not between successive steps. Memories and the separate saved Last X/Last Y operands remain intact; the examples do not depend on their previous contents.

3.2 Enter numbers, signs, and exponents

Type integers and decimals in their usual digit order. Use the decimal point key even if your regional writing convention uses a comma. Thousands separators appear automatically; do not enter them. A leading decimal point is sufficient for a number smaller than one: `.75` and `0.75` represent the same value.

To enter a negative number, type its magnitude and press +/− while **ENTRY** is showing. The subtraction key − performs an operation; it does not enter a minus sign. Pressing +/− again reverses the sign change. When no number is being entered, the same key negates the value already in X.

Try the independent rows of table 3.1, starting each from the prepared start. The final ENTER confirms the number, so the result uses FIX 4 rather than the unfinished entry's appearance.

Number	Keys after clearing	Display after ENTER
125	125 ENTER	125.0000
0.75	.75 ENTER	0.7500
−12.5	12.5 +/− ENTER	-12.5000

Table 3.1: Ordinary numeric entry in the chapter's prepared RPN state.

Powers of ten within a number

EEX introduces the exponent in a number of the form $a \times 10^b$. Enter the coefficient a, press EEX, then enter the integer exponent b. Do not type a multiplication sign or the number 10. For example, from the prepared start, f CLEAR 6.02 EEX 3 ENTER enters 6.02×10^3 and displays 6,020.0000.

While an exponent is active, +/− changes the *exponent's* sign. To make both the coefficient and exponent negative, change the coefficient's sign before pressing EEX. You can change the exponent's sign either before or after its digits.

Enter a small negative number

Use the prepared RPN start. Enter -4×10^{-3} with:

f CLEAR 4 +/− EEX +/− 3 ENTER

The entry passes through -4 E+ and -4 E-3; after ENTER, the display is -0.0040. The first sign change belongs to 4; the second belongs to the exponent. In the same prepared state, f CLEAR 2.5 EEX +/− 3 ENTER gives 0.0025.

If no entry is active, EEX starts a new number with coefficient 1. Thus, from the prepared start, f CLEAR EEX 3 ENTER gives 1,000.0000. If you want to multiply an existing result by a power of ten, enter that power as a new operand and press ×; EEX alone does not append an exponent to a completed result.

A second decimal point, a second EEX, or a decimal point within the exponent is ignored. An exponent with no digits is treated as zero when an RPN entry is accepted, so finish typing the intended exponent before pressing ENTER or a function. Backspace corrections are covered in section 3.8.

3.3 Add, subtract, multiply, and divide

For an ordinary two-number calculation, enter the first number, press ENTER, enter the second, and press the operator. The operator accepts the second number and immediately displays the result. No further ENTER is required.

For subtraction and division, enter the left-hand operand first: the calculator uses $Y - X$ or Y/X. Addition and multiplication use the same order, although reversing their two operands would give the same answer. In table 3.2, repeat the prepared start, including f CLEAR, before each row.

Calculation	Keys after clearing	Display
$27 + 15$	27 ENTER 15 +	42.0000
$52 - 19$	52 ENTER 19 −	33.0000
2.4×3	2.4 ENTER 3 ×	7.2000
$7.5/2.5$	7.5 ENTER 2.5 ÷	3.0000

Table 3.2: The four arithmetic operations from a cleared RPN stack.

Negative operands follow the same pattern. To calculate $-12.5 + 20$ from the prepared start, press f CLEAR 12.5 +/− ENTER 20 +. The display is 7.5000. The sign-change key belongs to the first number, while the final + combines the two numbers.

3.4 Squares, roots, reciprocals, and powers

Square, square root, and reciprocal need only X. For these real-number examples, use a nonnegative square-root input and a nonzero reciprocal input. Type the number and press the function; the function accepts the entry and replaces it with the result. Square root is a main key on the Scientific panel. Square, x^2, is that key's orange function. Reciprocal, $1/x$, is in the fixed control row.

A general power needs two numbers. Enter the base first, then the exponent, and press the Scientific panel's y^x key. Here Y is the base and X is the exponent. Table 3.3 gives complete examples after clearing; use the prepared start separately for each row.

You may apply another single-operand function directly to a result. For example, from the prepared start, f CLEAR 81 $\sqrt{x}$ f x^2 returns 81.0000. The result of square root becomes the operand of square.

A negative exponent is entered with the sign-change key just like any other negative number: f CLEAR 2 ENTER 3 +/− y^x gives 0.1250, since $2^{-3} = 1/8$. General roots, fractional powers, and other function domains are covered in chapters 7 and 8.

Calculation	Keys after clearing	Display
12^2	12 f x^2	144.0000
$\sqrt{81}$	81 $\sqrt{x}$	9.0000
$1/8$	8 $1/x$	0.1250
2^5	2 ENTER 5 y^x	32.0000

Table 3.3: Single-operand functions and a two-operand power in real RPN.

3.5 Continue from a result

A result is already a usable number in X. To combine it with a new operand, type that operand and press the next operator. The first digit lifts the previous result into Y. You do not need ENTER between a completed result and a new number; an extra ENTER there would duplicate the result.

Calculate a subtotal, then adjust it

Use the prepared RPN start, including FIX 4 and no pending command. Calculate $(18.50 + 7.25) \times 3 - 5$ in these successive steps. Continue straight down the table without clearing between rows:

Keys	Display
f CLEAR 18.50 ENTER 7.25 +	25.7500
3 ×	77.2500
5 −	72.2500

The first operation forms the subtotal, the second multiplies that subtotal, and the third subtracts 5 from the product.

The four displayed decimal places do not limit the value used in the next operation. For example, from the prepared start, f CLEAR 1 ENTER 3 ÷ shows 0.3333. Continuing with 3 × shows 1.0000, because the multiplication uses the internally retained quotient. Re-entering 0.3333 would instead use that shorter approximation. See chapter 5 for the difference between displayed digits and working precision.

3.6 Keep two intermediate results on the stack

When a calculation has two parts, complete one part and leave its result on the stack while you calculate the other. The new entries move the first result deeper; completing the second part brings the two results together in Y and X.

Divide one subtotal by another

Calculate

$$\frac{18+6}{9-5} = \frac{24}{4} = 6.$$

Use the prepared RPN start, real mode, fixed depth 5, FIX 4, and no pending shift or command. Press:

f CLEAR 18 ENTER 6 + 9 ENTER 5 − ÷

The final display is `6.0000`.

The numerator 24 waits while you enter 9 and 5. Subtraction consumes those two newer operands and leaves 4 in X, with 24 immediately above it in Y. The final division therefore uses 24/4. No memory register or parentheses key is needed.

Position	After +	After 9 ENTER	After 5	After −	After ÷
S4	0	0	0	0	0
S3	0	0	0	0	0
S2	0	0	24	0	0
S1/Y	0	24	9	24	0
S0/X	24	9	*5*	4	6

Table 3.4: The two subtotals through the five-register stack. Italic 5 is still being entered; subtraction accepts it before calculating. Values omit trailing decimal places.

You can pause after any step and open the full-stack inspector to check these positions; inspection does not finish an entry or move a value. If you accidentally calculate the denominator first, the final two results will be reversed. Use $x \leftrightarrow y$ before division to restore numerator in Y and denominator in X. For systematic ways to choose an evaluation order for larger expressions, continue with chapter A.

3.7 Use the percentage keys

The blue functions of 1, 2, and 3 are %, Δ%, and T%. All three use Y and X, but their stack behavior differs from multiplication or division: they replace X and leave Y and every deeper value in place.

Enter a rate of 15 percent as `15`, not `0.15`. Conversely, a percentage result of `15.0000` means 15 percent. Percentage change and percentage of a total require a nonzero original value or total.

Function	Enter first, then second	Result in X
g %	Base Y, percentage rate X	$YX/100$
g Δ%	Original value Y, new value X	$100(X-Y)/Y$
g T%	Total Y, part X	$100X/Y$

Table 3.5: The three percentage functions. The base, original value, or total remains in Y.

Add or subtract 15 percent of a price

Use the prepared RPN start, real mode and FIX 4, with no pending command. Find 15 percent of 80:

f CLEAR 80 ENTER 15 g %

The display is `12.0000`; Y still holds `80.0000`. Press + to add the increase and obtain `92.0000`.
To apply a discount instead, repeat the complete sequence from CLEAR, then press − in place of +. The result is `68.0000`. Both calculations reuse the retained base of 80; no second entry of 80 is needed.

Compare a new amount and find a share

Use the prepared RPN start for each of the following independent calculations, with no pending shift or command.
For the change from 80 to 92, press:

f CLEAR 80 ENTER 92 g Δ%

The display is `15.0000`: a 15 percent increase. Y remains 80. Replacing 92 with 68 in the same sequence gives `-15.0000`, a 15 percent decrease.
To find what percentage 50 is of a total of 200, press:

f CLEAR 200 ENTER 50 g T%

The display is `25.0000`, and Y remains 200. The answer is 25 percent, not an amount of 25 in the original units.

Retaining the base makes % followed by + or − convenient. It also means that a completed percentage calculation leaves an extra operand behind. Inspect or deliberately clear the stack before starting unrelated work. Do not assume that a percentage function drops the stack like an ordinary two-operand operation.

3.8 Correct an entry or recover operands

Choose a correction according to whether the number is still being entered, has already been accepted, or has been used by an operation. The **ENTRY** indicator distinguishes the first two cases.

Correct digits while entering

With no pending shift or command, ← removes the last entry character. For example, from the prepared start, suppose you type 12.36 but mean 12.35. The complete correction and calculation are:

f CLEAR 12.36 ← 5 ENTER 7.65 +

The result is 20.0000. Backspace removes only the 6; the following 5 edits the same entry.

Within an exponent, backspace removes exponent digits first. After the last exponent digit has gone, one more backspace removes the entire exponent marker and its sign, returning to the coefficient. For example, from the prepared start, f CLEAR 2.5 EEX +/− 4 ← 3 ENTER gives 0.0025. It corrects 2.5×10^{-4} to 2.5×10^{-3}. To change only an exponent's sign, use +/− while that exponent is still active.

Deleting every character ends a real-number entry and leaves zero in its reserved X position. It does not undo the lift that started the entry. Once **ENTRY** disappears, backspace acts as CLx; it no longer removes digits from the displayed value. See section 2.8 for that distinction.

Replace an accepted operand

If the wrong operand is already accepted, drop it before entering its replacement. DROP is the orange function of the roll-down key. This preserves the other operand that was waiting in Y.

For example, use the prepared start and deliberately accept 6 where you meant 7:

f CLEAR 84 ENTER 6 ENTER f DROP 7 ÷

The display is 12.0000. DROP exposes 84; typing 7 then lifts 84 back into Y, ready for division. The same replacement method works after you have erased an entry completely: drop the zero position, then enter the replacement.

Do not substitute CLx for DROP in this sequence. CLx leaves a zero in X; entering a new number would lift that zero into Y and move 84 another position deeper. CLEAR, by contrast, is useful when you intend to discard the whole calculation and start again.

Recover after the wrong operation

After a successful operation, LAST Y and LAST X can recover its saved operands. These saved values are separate from the current S1/Y and S0/X. Use them before another calculation or a clearing/recall command that changes those saved values. As explained in section 2.9, they recover operands rather than a complete earlier stack.

LAST Y is the blue function of $1/x$; LAST X is the blue function of $x \leftrightarrow y$. DROP and the two saved-operand recalls preserve both saved operands until the corrected operation uses them. Avoid CLx here: it saves the mistaken result as Last X, replacing the operand you need.

Repair one operation and keep an earlier subtotal

Use the prepared RPN start with fixed depth 5, real mode, FIX 4, and no pending command. You intend to calculate $(20 + 4) + (84/7)$, but press multiplication for the division:

f CLEAR 20 ENTER 4 + 84 ENTER 7 ×

The display is 588.0000. The earlier subtotal 24 is in Y; the saved operands are Last Y = 84 and Last X = 7. Immediately press:

f DROP g LAST Y g LAST X ÷ +

Dropping removes 588. The two recalls put 84 in Y and 7 in X, with 24 waiting in S2; division produces 12, and addition gives the intended 36.0000.

If the operation reported an error instead of producing a result, inspect the entry and stack before choosing a correction; do not assume that the preceding recovery sequence applies to every error. Chapter 14 covers error recovery.

3.9 A short practice set

Use the prepared start before each exercise: RPN, real mode, fixed depth 5, 600 working digits, FIX 4, no pending shift or command, and f CLEAR. Work out a sequence before looking at the answers. The settings and cleared start also apply to every answer sequence below.

1. Calculate $35 - 8$, then divide that result by 3.
2. Calculate $-6.5 + 2.25$.
3. Enter 3.2×10^{-3}, then multiply it by 250.
4. Calculate $\sqrt{144} + 1/4$.
5. Calculate $(15 + 9)/(8 - 2)$.

6. Calculate 3^4.
7. Reduce 120 by 12.5 percent.
8. Find the percentage change from 240 to 270.

Answers and one possible sequence

In table 3.6, each sequence starts *after* f CLEAR. Line breaks within an answer are only for readability; do not clear between the parts of a sequence.

Table 3.6: Practice answers in FIX 4.

No.	Keys after clearing	Display
1	35 ENTER 8 − 3 ÷	9.0000
2	6.5 +/− ENTER 2.25 +	-4.2500
3	3.2 EEX +/− 3 ENTER 250 ×	0.8000
4	144 $\sqrt{x}$ 4 $1/x$ +	12.2500
5	15 ENTER 9 + 8 ENTER 2 − ÷	4.0000
6	3 ENTER 4 y^x	81.0000
7	120 ENTER 12.5 g % −	105.0000
8	240 ENTER 270 g $\Delta\%$	12.5000

For exercise 4, square root leaves 12 waiting while reciprocal produces 0.25; addition combines them. For exercise 7, the percentage amount is 15 and the retained base is 120, so subtraction gives 105. For exercise 8, the increase is 30 on an original 240, giving $100 \times 30/240 = 12.5$ percent. These intermediate checks help locate an incorrect key before you repeat a whole calculation.

3.10 Where to go next

If the results are correct but the operand movements still feel unfamiliar, revisit chapter 2 with the full-stack inspector open between steps. For longer expressions and more practice, continue with chapter A. Chapter B discusses the reasons behind this style of calculation.

For everyday use, chapter 5 explains the digits you see, chapter 6 shows how to keep reusable values, and chapter 7 extends these basic functions. If you also want to enter written expressions, the next chapter introduces the independent ALG workspace.

Chapter 4

Calculations in ALG Mode

In ALG, you enter an expression in its written order and press = to evaluate it. Parentheses identify groups, and precedence determines which operations are carried out first. You can enter a function before its argument, apply it to an operand already entered, or reuse the answer to an earlier calculation.

ALG has its own working area. It does not use the RPN stack, and selecting ALG does not transfer the RPN display into an expression. The scientific keyboard, ordinary memories, and settings are shared; the available calculations are summarized in section 4.10.

4.1 Entering and evaluating an expression

Select ALG above the display and show the Scientific panel. The Expression row contains (,) , FUNC , CPT , and CLEAR . In portrait it is above the Shift and Memory row: f , g , ANS , STO , and RCL . In landscape Expression sits below the Scientific and Financial panels on the left, while Shift and Memory sit above entry on the right. The left entry column has normal-size EEX , $1/x$, +/− , and = keys from top to bottom. The = key is on the bottom row. Orange $1/x$ selects INT, which truncates toward zero; blue $1/x$ selects FRAC, which returns the signed fractional part. Equals, ANS, and CPT have no shifted legends. CPT selects an explicit financial computation, described in section 4.11. The bottom line shows the expression in both real and complex mode and is available to VoiceOver. Calibration appears above it when digits overflow; drag the digits to scroll. Prompts and errors appear below the flags, above the numbers. The large number, or the two complex component rows, shows the number being entered or the most recent operand or function preview. It is not a running total of an unfinished expression.

Numbers in the expression line use four places after the decimal point, independently of the large display's digit count. The selected FIX, SCI, or ENG notation still applies. In FIX, a number that needs more than four integer digits switches to scientific notation, also with four decimal places: for example, 12345 appears as `1.2345E+4`. Very small nonzero values also use scientific notation if FIX 4 would round them to zero. This compact presentation applies to both complex components and to complete numbers still being typed. The large display keeps the exact entry text, and unfinished entries such as `3 E-` retain their editing cues in the expression line. Only the presentation is rounded; calculations use the full operands.

The **ENTRY** indicator remains on while an expression is unfinished, including between numbers and after a closing parenthesis. Pressing = evaluates the whole expression, displays its result, and turns off **ENTRY**. The completed expression remains visible until you start or continue a calculation.

Before each independent example, make sure no shift, command, or digit-count prompt is pending, then press the unshifted **CLEAR** key. This clears the ALG working expression without clearing its previous answer or any memory bank. Prefix cancellation is explained in section 1.8; clearing in ALG is covered in section 4.9.

For the examples in this chapter, use real mode (**COMPLEX** off), **DEG**, FIX 4, and 600 working digits. These settings make the displayed answers easy to compare. Check working digits with **PREC**, as described in chapter 5. In ALG, select FIX 4 by pressing **f** **7** 4 =. The orange legend on **7** is **FIX**; this particular equals press accepts the display setting and does not evaluate an expression. Set degrees with **f** **SIN**, selecting its orange **DEG** legend.

Evaluate a first expression

Use ALG, real mode, **DEG**, FIX 4, and 600 working digits, with no pending shift or command. Press:

CLEAR 2 + 3 × 4 =

The result is `14.0000`, because multiplication is performed before addition: $2 + (3 \times 4) = 14$. Before the final =, the large display shows the final entry, 4; use the expression line to check the complete calculation.

Enter integers and decimals with the digit keys and decimal point. To make a number negative, enter its digits and press +/−. For scientific notation, enter the mantissa, press **EEX**, then enter the exponent. While the exponent is active, +/− changes the exponent's sign rather than the mantissa's sign.

For example, from the chapter's cleared ALG starting state, `1.25` **EEX** +/− 2 + `.5` = gives `0.5125`. The calculation is $1.25 \times 10^{-2} + 0.5$. If **EEX** starts a new number, the mantissa is supplied as 1: **EEX** 3 = gives `1,000.0000`. An exponent needs at least one digit before evaluation; 2 **EEX** = does not mean 2×10^{0}.

Physical-keyboard shortcuts also enter digits, operators, parentheses, and commas. Return evaluates, `^` selects the power key, `e` selects **EEX**, and `s` selects +/−. These are key shortcuts, not a general text editor for function names or pasted formulas. Use the visible function keys and **FUNC** for the procedures below. See table 1.4 for the common shortcuts.

4.2 Precedence and parentheses

Multiplication and division take priority over addition and subtraction. Operations at the same priority are normally evaluated from left to right. Thus $8/4/2 = (8/4)/2 = 1$, and $10 - 3 - 2 = 5$. Powers are the exception: a chain of powers is evaluated from the right, as explained in section 4.3.

Parentheses make a group into one operand. Use them whenever the intended grouping would otherwise be unclear. The expression $(2 + 3) \times 4$ gives 20, whereas $2 + 3 \times 4$ gives 14. Every opening parenthesis needs its own closing parenthesis; `=` does not supply missing ones.

Table 4.1: ALG binding order, from tightest to loosest.

Construct	How it is grouped
Groups and function calls	Parentheses delimit the expression or arguments that belong together.
Functions after an operand	A unary function applies to the preceding number, closed group, or function result.
Powers	$a^{b^c} = a^{(b^c)}$.
Leading mathematical minus	Applies after a following power: $-2^2 = -(2^2)$.
Multiplication-level operations	Multiplication, division, and the other binary scientific keys except power; equal priority is resolved from left to right.
Addition and subtraction	Equal priority is resolved from left to right.

The multiplication-level scientific keys include indexed roots, base logarithms, permutations, combinations, **atan2**, the two remainders, and the three percentage operations. For a complicated expression involving these keys, an explicit **FUNC** call makes the two arguments clearer than a long chain of binary operations. Implicit multiplication has the same priority as ordinary multiplication; it does not receive extra priority because the multiplication sign was omitted.

Group two intermediate calculations

Use ALG, real mode, **DEG**, FIX 4, and 600 working digits, with no pending shift or command. Enter:

CLEAR (8 + 4) ÷ (5 − 2) =

The numerator is 12 and the denominator is 3, so the result is `4.0000`. Closing the denominator shows its value, 3, as a preview; only the final `=` evaluates the full quotient.

4.3 Powers and negative operands

The y^x key enters a power between its base and exponent. With no parentheses, 2^{3^2} means $2^9 = 512$, not $8^2 = 64$. To square the result of 2^3, enclose that first power in parentheses.

There are also two distinct ways to enter a negative operand. Pressing the subtraction key where an operand is expected inserts mathematical minus. It has lower priority than power. Changing the sign of a number while typing its digits makes the sign part of that number instead. The expression display puts a committed negative number in parentheses to make this distinction visible.

Table 4.2: Powers and signs. For each row, use the chapter's real ALG, DEG, FIX 4, 600-digit settings, with no pending prefix, and first press **CLEAR**. Results omit four trailing decimal places.

Keys after clearing	Result
2 y^x 3 y^x 2 =	512
(2 y^x 3) y^x 2 =	64
− 2 y^x 2 =	−4
2 +/− y^x 2 =	4
2 y^x 2 +/− =	0.25

The same distinction applies to +/− when no number is being typed. At an empty operand position it inserts a leading minus; after an existing operand it negates that operand. During exponent entry it changes only the exponent's sign. For an unmistakable negative base, enter the signed number first, or enclose the negative expression in parentheses before pressing y^x.

4.4 Implicit multiplication

ALG supplies multiplication in several useful positions. An opening parenthesis after an operand starts a new factor. A number immediately after a closing parenthesis is another factor, as is a second group. A constant key or an explicit **FUNC** call after an operand also inserts multiplication. Selecting **Re/Im** to edit an existing complex operand is different: it replaces the trailing group or function value while preserving the surrounding expression. See section 8.2.

For example, with the chapter's cleared real ALG starting state, 2 (3 + 4) = gives `14.0000`: it means $2 \times (3 + 4)$. The expression line includes the inserted multiplication sign.

The orange function of **0** is π, so the last row calculates 2π. This is display rounding of the result, not a four-place approximation used internally.

Table 4.3: Other implicit products. Use the same settings and no pending prefix; press CLEAR before each row.

Keys after clearing	Display
(2 + 3) 4 =	20.0000
(2 + 3) (4 + 1) =	25.0000
2 f 0 =	6.2832

Do not extend this rule to every pair of adjacent key presses. Digits typed consecutively form one number: 2 3 enters 23. A unary function after a number acts on that number, as described next. ANS and ordinary memory recall can replace a trailing operand. Use an explicit × before either when you want a product.

4.5 Functions before and after an operand

A single-operand function can be entered in two orders. If an operand is expected, pressing its function key opens a function call and its opening parenthesis. Enter the argument and close the parenthesis. At the start of an expression, for example, $\sqrt{x}$ 9) = calculates $\sqrt{9}$.

If an operand is already present, pressing the same function key applies it to that operand. This is called *postfix* use: 9 $\sqrt{x}$ = also gives 3. The expression line marks a postfix function with square brackets, such as `[sqrt]`. Those brackets describe the function just selected; they are not extra keys to enter.

The affected operand is the immediately preceding number, closed group, or function result. A postfix function does not automatically apply to everything to its left. In $2 + 9$, square root after 9 produces $2 + \sqrt{9} = 5$. To take the square root of the sum, close $(2 + 9)$ first and then press square root. Likewise, in 2^3, square root pressed after the 3 applies to the exponent; use (2^3) if the entire power is the intended argument.

Choose the angle unit before inserting a trigonometric function. The function entered into an expression retains the unit selected at that key press; changing the displayed angle setting afterward does not rewrite functions already entered. The expression's function label identifies its unit. Hyperbolic functions do not use angle units. Function domains and complex extensions are covered in chapters 7 and 8.

Successive postfix unary functions act in the order pressed. Shifted functions follow the same placement rule: for example, the orange x^2 function on the square-root key squares the preceding operand. HYP followed by SIN, COS, or TAN selects a hyperbolic function; the resulting function opens a call when an argument is needed or acts on the preceding operand. Use orange HYP for the corresponding inverse-hyperbolic selection.

Table 4.4: Function placement. Use real ALG, DEG, FIX 4, and 600 working digits, with no pending prefix; press CLEAR before each row.

Keys after clearing	Display
$\sqrt{x}$ 9) =	3.0000
2 + 9 $\sqrt{x}$ =	5.0000
(2 + 7) $\sqrt{x}$ =	3.0000
SIN 30) =	0.5000
30 SIN =	0.5000

4.6 FUNC and multiple arguments

FUNC explicitly selects a function call before its arguments. Press FUNC, then a unary or binary mathematical function key. The **FUNC** indicator appears while the calculator waits for that selection. The selected function and its opening parenthesis are then inserted together; do not add another opening parenthesis unless you need a nested group within the argument.

Shifts can be used between FUNC and the function key. For example, FUNC g y^x selects the blue base-logarithm function. Constants and random numbers are values, so they do not use FUNC. HYP is itself a prefix rather than a mathematical function key: use the ordinary HYP sequence described above, without a preceding FUNC.

A unary function takes one argument. A binary function takes two, separated by a comma. Enter the comma with g FUNC, the blue legend on that key; a physical comma key does the same thing in ALG. A comma is not a thousands separator or a decimal point. It belongs inside a two-argument function call, and each argument can itself be a complete expression. Close the call with), then continue the surrounding expression or press =.

The left argument of a binary function corresponds to the left operand of its ordinary ALG form. For functions also described in RPN notation, it is the Y (S1) operand; the second argument is the X (S0) operand. These are argument positions here, not registers in an ALG stack.

Enter a base logarithm with two arguments

Use ALG, real mode, **DEG**, FIX 4, and 600 working digits, with no pending shift or command. Enter $\log_2 8$ with the base first and the argument second:

CLEAR FUNC g y^x 2 g FUNC 8) =

The expression uses the label `logbase`; the result is `3.0000`, since $2^3 = 8$.

Table 4.5: Argument order for representative binary functions.

Function key after FUNC	First argument	Second argument
y^x	Base	Exponent
f y^x	Radicand	Root index
g y^x	Logarithm base	Argument
g $x!$	Number of objects, n	Number selected, r, for nCr
g Γ	Number of objects, n	Number selected, r, for nPr
g HYP	Vertical coordinate, y	Horizontal coordinate, x, for atan2(y, x)
g 1	Base amount	Percentage rate
g 2	Original amount	New amount
g 3	Total amount	Part of the total

For a second two-argument example, use the same cleared starting state and enter FUNC f y^x 81 g FUNC 4) =. This calculates the fourth root of 81 and displays 3.0000.

After an existing operand, FUNC starts a new factor. Thus 2 FUNC $\sqrt{x}$ 9) =, from the same cleared state, gives 6.0000: it means $2\sqrt{9}$. Without FUNC, square root after 2 would act on 2 instead.

Percentage keys in an expression

The blue 1, 2, and 3 functions are two-operand operations, not a percent suffix on a single number. For example, from the chapter's cleared starting state, 200 g 1 10 = gives 20.0000, which is 10 percent of 200. The equivalent call is FUNC g 1 200 g FUNC 10) =.

To add that amount to the base, enter it explicitly, for example $200+200\times10/100$, or continue from the percentage result with + 200 = to get 220.0000. ALG has no retained Y stack operand to supply the base automatically. Use the percentage keys for these functions; the MOD and REM keys are separate remainder operations, explained in chapter 7.

4.7 Editing an unfinished expression

Before editing, dismiss any pending shift or command with the header's CANCEL. This preserves the expression; an unfinished expression alone does not make CANCEL

appear. With no command pending, the editing keys work as described below. For alternate cancellation methods and their exceptions, see section 14.4. If a worksheet is open, follow its separate editing and cancellation rules in section 4.12.

Editing works at the right-hand end of the expression. While a number is being typed, ← removes its last character. It removes exponent digits before leaving exponent entry, and removes fractional digits before the decimal point. Deleting the entire number removes that operand; it does not insert a zero into the expression. The large display may show zero while it waits for a replacement.

When no number is being typed, backspace removes the last expression element, such as an operator, a closing parenthesis, or a postfix function. Removing a function's opening parenthesis also removes its function name. A committed number is removed as one element, not reopened for character-by-character editing. There is no movable insertion cursor for selecting an earlier digit: delete back to the required position or re-enter the expression.

CLx, the blue function of backspace, removes the current entry or the trailing operand. A closed group or function call, together with its postfix functions, counts as one operand. The earlier expression remains. If the expression already ends with an operator, **CLx** leaves that operator waiting for its operand. It does not replace a missing operand with a literal zero.

Replace a mistyped operand

Use ALG, real mode, **DEG**, FIX 4, and 600 working digits, with no pending shift or command. Suppose you intend $12 + 35$, but type 36. Correct its final digit with:

CLEAR 12 + 36 ← 5 =

The result is `47.0000`. To replace the whole second number instead, repeat from **CLEAR** and use:

12 + 36 g ← 35 =

The blue backspace selects **CLx**; the result is again `47.0000`.

You can also correct an operator before entering its right operand. A new binary operator replaces a preceding binary operator, except that subtraction in an operand position introduces a negative operand. From the chapter's cleared starting state, 2 + × 3 = gives `6.0000`. But 2 × − 3 = gives `-6.0000`. To replace a mistaken operator with subtraction, remove it with backspace first, then press subtraction.

Once a trailing number or postfix result has been committed, new digits replace that trailing operand. They do not append digits to its displayed result. A closing parenthesis is different: a number typed immediately after it starts an implicit

product. For a reliable replacement of a closed group, press CLx first, then enter the new operand.

4.8 ANS, continuation, and ordinary memory

ANS supplies the saved ALG answer. It starts at zero in a new calculator state. Successfully evaluating the main expression with = updates it; a failed evaluation or CLEAR does not. It is independent of the RPN result and RPN's saved operands, Last X and Last Y. Financial computations and worksheet actions can also update ANS, as explained in sections 4.11 and 4.12.

After a completed result, a binary operator continues from that result. A unary function can also act on it. New digits, a decimal point, or EEX instead begin a new expression. Opening a parenthesis, selecting an explicit FUNC call, inserting a constant, or recalling a value also begins new work after a completed result. To include the previous result in such a new expression, insert ANS at the appropriate operand position.

Repeated = leaves a completed result unchanged and records that result as ANS again. It does not repeat the previous arithmetic operation. Also distinguish a function preview from a completed answer: after applying a unary function to a result, press = to make the new value the saved answer.

Continue a result, then reuse ANS

Use ALG, real mode, **DEG**, FIX 4, and 600 working digits, with no pending shift or command. Start with:

CLEAR 2 + 3 =

The answer is 5.0000. Continue with × 4 = to obtain 20.0000. Pressing = again still shows 20. Now press:

CLEAR 100 ÷ ANS =

CLEAR preserved the answer 20, so the new result is 5.0000. Typing 7 = now starts a new calculation and makes 7.0000 the answer.

Within an unfinished expression, ANS inserts a value where an operand is needed. If an operand is already present at the end, it replaces that operand, including a trailing closed group or function call. For example, to multiply a typed 2 by the previous answer, press 2 × ANS; pressing ANS directly after 2 would replace the 2. The value inserted is the answer available at that moment.

With CMPLX off, inserting ANS copies only its real component into the expression and leaves the saved answer unchanged. A later successful evaluation replaces ANS

with its new result. Merely toggling CMPLX or reopening the application preserves the saved answer and expression operands.

Ordinary RCL followed by a digit uses the same insertion and replacement rule. STO followed by a digit stores the current operand or preview without evaluating the entire unfinished expression. Both commands finish a number that is still being typed when the register is selected. To store the result of a whole expression, press = before STO.

For example, use the chapter's settings and no pending prefix. This example deliberately replaces memory R0 with 3; the register need not have been empty. Press CLEAR 2 + 3 STO 0 =. The expression evaluates to 5.0000. Confirm the stored operand with CLEAR RCL 0 =: the display is 3.0000. For arithmetic storage and the shared memory banks, see chapter 6.

4.9 Clearing and recovering from an error

With no pending shift or command, the unshifted CLEAR key discards the whole ALG expression and any unfinished number, returning the display to zero. It preserves ANS, memories, settings, and the RPN workspace. CLx removes only the current entry or trailing operand. After a completed result, either CLx or backspace clears the completed working expression; neither changes the saved answer.

An incomplete expression is retained when evaluation fails. Read the message, then supply the missing item or correct the operand. The last successful ANS remains available. A valid number may have been committed before the error was detected, so subsequent backspace can remove a complete number rather than one digit.

Table 4.6: Repairing common ALG errors. Use the chapter's settings and no pending prefix; begin each row with CLEAR.

Enter	Repair and expected display
2 + =	An operand is missing. Enter 3 = to get 5.0000.
(2 + 3 =	Close the group with) = to get 5.0000.
2 EEX =	The exponent is incomplete. Enter 3 = to get 2,000.0000.
1 ÷ 0 =	The expression has no finite result. Press g ← 2 = to replace the denominator and get 0.5000.

A misplaced closing parenthesis, a comma outside a binary function call, or the wrong number of arguments is rejected. The failed structural change is not added to the expression. For example, if you try to close a two-argument call after its first

argument, enter the comma and second argument before closing it again. An invalid postfix function similarly leaves its operand available for correction.

Real mode cannot return a finite real answer for expressions such as $\sqrt{-1}$ or division by zero. Complex mode permits some additional function domains, but does not make every undefined calculation valid. Consult chapter 8 before using complex entry to address a domain error. Expressions also have limits of 4096 stored elements and 128 nested groups; an individual numeric entry is limited to 10 000 characters. If a limit is reached, split the work into smaller calculations and save useful results in memory.

4.10 Switching workspaces and choosing available functions

Switching modes preserves each workspace, including the ALG expression, unfinished number, completed-result state, and answer. It does not evaluate, finish entry, or transfer operands. To transfer a value, store it in ordinary memory, then recall it in the other mode.

Mode selection cancels shifts and pending prefixes. With an ALG **ΔFIX** prompt, the first mode-button press only cancels that prompt; press again to switch.

Precision, angle and display settings, complex mode, and memory banks are shared. Both workspaces survive relaunches; see chapter 14. The table below identifies available functions. See chapter 15 for individual commands' keys and operands.

Table 4.7: Choosing an entry mode for the current keyboard.

Kind of work	Availability and further reading
Arithmetic and scientific keys	Both modes, including shifted scientific functions and percentages. See chapter 7.
Ordinary memory	Both modes; insertion and replacement in ALG differ from RPN stack recall. See chapter 6.
Complex values	Both modes, with component editing and rectangular/polar presentation. See chapter 8.
Precision, notation, and conversions	Both modes. FIX has a different entry procedure in each. See chapters 5 and 9.
Statistics and associative memory	Both modes: stack operations in RPN, labeled worksheets in ALG. See section 4.12 and chapters 6 and 10.
Financial and date calculations	Both modes: explicit CPT and worksheets in ALG. See chapters 11 to 13.

Flags such as **STAT**, **FIN**, and **ASSOC** report shared occupied banks. Changing an input in either mode changes that shared bank; neither mode uses the other mode's working operands.

4.11 Financial assignments and explicit computation

On the Financial panel, the unshifted n, i, PV, PMT, and FV keys always store the displayed operand. They never infer a solve from the preceding action. Explicit STO and arithmetic STO can also target these registers. Recall with RCL followed by the financial key; it inserts or replaces an operand just like ordinary memory recall.

CPT is a pending prefix: the display identifies the required target (a financial register, NPV, or IRR). CANCEL dismisses it; relaunch restores the selection without solving anything. A direct solve requires a completed or cleared main expression. If a compound expression is unfinished, the message `Finish the expression first` leaves it intact. A lone entered scalar does not block computation. A successful TVM solve stores the target, displays a completed result, and updates ANS. Direct NPV or IRR also displays a completed result and updates ANS, without overwriting a financial register. Digits start new work; an operator continues from that result.

The inputs retain SpicyRPN's existing definitions. In particular, TVM i is a percentage per payment period, not an annual I/Y coupled to P/Y and C/Y settings. Solved period counts retain the rounding described in section 11.10; ALG changes the entry procedure, not that mathematics.

Sequence	Meaning
250 PV	Store 250 in PV.
200 + 50 PV	Store 50; retain the pending addition.
200 + 50 = PV	Store the completed total, 250.
RCL PV	Insert the saved present value.
CPT PMT	Solve for PMT using the four other financial registers.

4.12 Working with labeled worksheets

Use a worksheet when an operation needs several inputs or has several outputs. Orange FUNC, labeled STAT, opens statistical worksheets; orange STO and RCL open associative storage and recall. Financial operations such as AMORT, bond price, cash-flow entry, and date arithmetic open their named worksheets. Each worksheet explains its inputs and uses the current angle, date, precision, and complex settings where relevant.

The main expression and its exact unfinished entry are suspended while a worksheet is open. The Operation selector switches among worksheets in the same category. On first use, fields contain displayed starting values or defaults; changing a field marks it unaccepted. Financial fields refresh from the shared registers when opened or after a computation, unless you have an unaccepted edit in that field. The fields

are separate text expressions, so you can enter arithmetic such as `12*30` within a field. They use a decimal point and parentheses; for example, `(12+3)*2` is 30. The worksheet uses text-field input, not the main calculator keyboard. Select a field and edit its text, then use these controls:

= Evaluate the selected field and update ANS. This previews a value; it does not accept it as an input or write its mapped register.

ENTER
Evaluate and accept the selected field, updating ANS. Return while editing a field performs this same acceptance. A field labeled with a financial register also stores that register. Other accepted inputs are retained in their worksheet.

COMPUTE or a named action
Unaccepted edits must be accepted or discarded before computation; the calculator does not silently calculate from stale inputs. Use the accepted inputs. STORE, APPEND, REPEAT, ADD SAMPLE, REMOVE SAMPLE, and RECALL describe the corresponding operation. All outputs and bank changes are validated before the operation writes any bank.

Use value
Insert the selected labeled output into the suspended expression using RCL's insertion/replacement rule, then close the worksheet. This transfer does not itself change ANS or evaluate the main expression.

CANCEL
In the worksheet or header, close the worksheet, discard unaccepted field edits, and return to the unchanged main expression. Earlier accepted values, ANS updates, and completed operations remain; cancellation is not undo.

Outputs remain individually labeled. For example, amortization returns Interest, Principal, and Balance; regression returns Slope and Intercept, with Correlation when defined. The declared primary result updates ANS after successful computation. Selecting another output for inspection does not update ANS. Results describe the last successful computation; editing inputs does not recompute them. Recomputing uses the accepted inputs and the current relevant bank; simply reopening or browsing a worksheet does not execute it.

Field evaluation and acceptance can change ANS before you compute, as described above. A failed action or output browsing does not change it, and CANCEL does not restore an earlier answer. If a worksheet has changed ANS, pressing `=` again on a completed main expression records that expression's result as ANS again.

Worksheet state is saved with each action: its identity, accepted inputs, exact unfinished text, selected field, outputs, and selected output. Saved worksheet values retain their numerical components and precision, without retaining the full calculation history that produced them. Relaunch resumes an active worksheet without evaluating or accepting its draft. CANCEL then abandons only its unaccepted edits. In particular, accepting X for a statistical sample does not add half a sample: none

of the six accumulators changes until both coordinates and the complete addition or removal validate successfully.

See sections 6.7, 10.10, 11.12, 12.9 and 13.12 for the individual fields, outputs, and bank effects.

Complex operands

Complex ALG keeps the expression visible alongside the two component rows. **Re/Im** edits the trailing operand, including the value of a closed group or function call. It preserves the surrounding expression and leaves ANS unchanged until successful evaluation. See section 8.2 for an example.

Part II

Everyday and Scientific Use

Chapter 5

Numbers, Precision, and the Display

A calculator can retain many more digits than it shows. In SpicyRPN, working precision determines the precision available to new calculations, while the display setting determines how a value is presented. Neither setting establishes the accuracy of the numbers you supplied. Keeping these three ideas separate helps you read results, preserve useful intermediate values, and recognize when more digits would help.

The precision and display settings are shared by RPN and ALG. Their entry procedures differ in one important place: **FIX** takes a count from the stack in RPN, but opens a separate count prompt in ALG. This chapter covers both procedures.

5.1 Three meanings of precision

Working precision describes the numerical resolution used to create a value or calculate a result. SpicyRPN starts with 600 decimal working digits; **PREC** allows 16 through 10 000. Internally, the calculator uses binary floating-point values with enough bits for the selected decimal precision. For example, 600 decimal working digits correspond to 1 994 bits. This is a significant-digit precision, not a fixed number of decimal places: very large and very small numbers can use the same working precision.

Finite binary precision cannot represent every decimal or fraction exactly. Integers that fit the available precision, and fractions such as $1/8 = 0.125$, can be exact. A decimal such as 0.1, a repeating fraction such as $1/3$, and an irrational number such as π require approximations. Individual MPFR real-number operations use rounding to nearest, with an exact halfway case rounded to the even choice. A calculation made from several operations can accumulate or amplify their rounding errors.

Display precision controls the number of places shown after the decimal point, either in the ordinary number or in the coefficient of a power of ten. The initial **FIX 4** normally shows four decimal places. A short readout does not discard the remaining digits of the value. Likewise, requesting a long readout does not prove that all its digits are accurate.

Input accuracy comes from the source of the number. A length reported as 12.3 cm may be a rounded measurement; the calculator cannot discover the omitted

measurement detail. An exact count of 123 objects is a different kind of input. Entering extra zeros, increasing working precision, or displaying more digits cannot turn an uncertain measurement into an exact one. SpicyRPN does not infer measurement uncertainty from the number of digits you type.

Working digits are not guaranteed answer digits

The working setting is an arithmetic resource, not a promise of that many correct digits in every answer. Subtracting nearly equal numbers, using approximate inputs, evaluating a sensitive formula, or using an iterative or complex-number formula can reduce the useful accuracy. Choose enough working digits for the calculation, then judge how many answer digits the inputs and method justify.

5.2 Setting working and display digits with PREC

With no pending command or active shift, press g 7, the blue **PREC** function. The Precision & display panel shows the current settings in two typed fields:

Table 5.1: The fields in the precision panel.

Field	Accepted values	Effect
Working digits	Whole numbers 16–10 000	Precision for newly committed numbers and subsequent calculations.
Display digits	Whole numbers 0 through the proposed working count	Decimal places used by the selected FIX, SCI, or ENG format.

If you lower Working digits, also choose a Display digits count no larger than that new setting. For example, working 80 with display 81 is rejected. Raising the working count alone does not automatically raise the display count. The panel retains the selected notation, so applying new counts while **ENG** is showing leaves ENG selected.

Replace the field contents with the desired counts, then press **Apply**. The panel closes after a successful change. An invalid count leaves it open with a message so that you can correct the fields; it does not partly apply the new settings. Press **Close** to leave without applying the edited counts. The fields are settings, not operands: typing in them does not put numbers on the RPN stack or into an ALG expression.

Opening or applying **PREC** does not finish a pending numeric entry or evaluate an unfinished ALG expression. The characters already typed remain available for

editing. A numeric entry that is still open will be converted to a stored number at the working precision in effect when you finish it. In an ALG expression, numbers already finished before the change retain their earlier precision; changing the setting does not re-create them from their original text.

Change precision without finishing an entry

Use RPN, real mode, a fixed stack of 5, and FIX 4, with no pending shift or command. Press f CLEAR (the orange function of $1/x$), then type `1.2300` without pressing ENTER.
Press g 7. Set Working digits to 80 and Display digits to 8, then press Apply. The main number remains the entry `1.2300`, and **ENTRY** remains visible. Now press ENTER. The entry finishes and the display becomes `1.23000000` under **FIX 8**. The number was committed at the new working precision; the extra display zeros do not represent additional input information.

The INFIX label

The calculator's help uses **INFIX** as a name for working precision, including the upper bound for a display-digit count. In this usage, "current INFIX precision" means the value of Working digits in PREC. It does not select or identify algebraic entry. The entry-mode buttons above the display say RPN and ALG; those are the controls to consult when choosing a key sequence. Working precision applies in either mode.

5.3 Fixed, scientific, and engineering notation

The display's notation flag combines a format and a count, such as **FIX 4**, **SCI 4**, or **ENG 4**. The count means places *after the decimal point* in all three formats. It does not mean the same total number of significant digits in each format. The ALG expression line has its own compact four-decimal presentation, described in section 4.1; changing the main display's count does not lengthen the numbers in that line.

With a count of zero, the coefficient has no fractional part or decimal point. SCI still has one significant digit, while ENG can have one, two, or three. Zero itself is shown as an ordinary zero in FIX and with exponent zero in SCI or ENG, with the requested fractional zeros.

Press f 8, the orange FSE function, to cycle FIX to SCI to ENG and back to FIX. Each press sequence advances one step and retains the count. It does not calculate, move stack values, finish an entry, or evaluate an expression. While you

Table 5.2: What a display count of d means.

Format	Presentation
FIX	An ordinary decimal with d fractional places. The number of significant digits depends on its magnitude. Scientific notation is used when necessary, as explained below.
SCI	One digit before the decimal point and d after it, followed by a power of ten: $d + 1$ significant digits for a nonzero value.
ENG	One, two, or three digits before the decimal point and d after it, followed by a power of ten whose exponent is a multiple of three: $d + 1$ through $d + 3$ significant digits for a nonzero value.

are typing a number, the display continues to show the entry as typed; the selected format applies to the completed value.

Show one value in all three formats

Use RPN, real mode, a fixed stack of 5, and 600 working digits, with no pending shift or command. Press:

f CLEAR 12345.5 ENTER 2 f 7

The last three presses select **FIX 2**, using the RPN procedure explained in section 5.4. Continue as follows:

Action	Flag	Readout
Starting result	**FIX 2**	`12,345.50`
f 8	**SCI 2**	1.23×10^4
f 8	**ENG 2**	12.35×10^3
f 8	**FIX 2**	`12,345.50`

The stored value remains 12345.5 throughout. The main display presents the power of ten with a raised exponent; a text form such as `1.23E+4` has the same meaning. On a very narrow display, the FIX row may use scientific fallback.

Why FIX sometimes shows an exponent

FIX is a preference for ordinary decimal notation, not a requirement to show every number that way. If the integer portion, sign, grouping, and decimal boundary cannot fit the main display's width, the display uses scientific notation. Rounding can itself cause the integer portion to grow: a value close to 999.999 may round up to 1,000.00. Resizing a window can therefore change whether a particular value uses an exponent.

FIX also uses scientific notation when ordinary rounding would turn a nonzero value into displayed zero. For example, 0.000001 under FIX 2 appears as 1.00×10^{-6}, not `0.00`. An actual zero still displays as `0.00`. This helps distinguish a small retained result from zero, though it cannot recover a value that arithmetic has already rounded to zero.

The flag remains FIX with its selected count during fallback. The scientific coefficient uses that many fractional places. Long fractional tails remain available by scrolling; SpicyRPN does not reduce the requested number of decimal places merely to squeeze them onto one screen. A secondary line or expanded-stack row can have a different width from the main display, so judge the value together with its exponent rather than expecting identical line breaks or notation in every location.

5.4 Setting FIX in each entry mode

FIX is the orange function of **7**. It both selects FIX notation and sets its decimal-place count. The accepted count is a whole number from zero through the current working precision. To change the count while keeping SCI or ENG selected, use **PREC**; alternatively, set FIX and then cycle with **FSE**.

RPN: enter the count first

In the RPN stack, X is S0, the displayed operand, and Y is S1, the next position. Entering the count follows the normal stack-lift rule. On a full fixed stack, that lift discards the deepest value; the subsequent drop repeats the new deepest value. Thus the sequence returns the previous X to the display, but does not necessarily restore every deeper register. Use the **PREC** panel when you want to change display digits without using a stack position. FIX itself leaves the separately saved Last X and Last Y operands unchanged.

In RPN, enter the count and then press **f** **7**. For example, 6 **f** **7** selects FIX 6. There is no digit prompt after the FIX key. It reads the count already in X, finishes that entry if necessary, and drops it. The preceding Y becomes the displayed X. You do not need **ENTER** between a freshly typed count and FIX.

An invalid count, such as 1.5, a negative integer, or a number above the working precision, reports an error and leaves the setting and operand intact. If the count was still being entered, its entry remains editable. Correct the count before trying FIX again.

Set RPN decimal places and retain the displayed value

Use RPN, real mode, fixed depth 5, and 600 working digits, with no pending command or shift. Press:

f CLEAR 42 ENTER 2 f 7

The count 2 is consumed and the display reads 42.00. Now press 0 f 7. The same value appears as 42 under **FIX 0**. There is no fractional part in the readout, but this setting does not turn other stored values into integers.

ALG: select FIX, enter the count, then accept it

In ALG, press f 7 first. The Δ**FIX** flag and a status prompt identify a separate digit-count entry. Type the count, then press =. The count does not enter the expression, and this = accepts the display setting without evaluating the expression. Both an unfinished number and the rest of the expression resume exactly where you left them.

Only digits and the accepting = belong to this prompt. Any other key cancels it and is consumed as cancellation; it does not also perform that key's normal function. In particular, backspace cancels the prompt rather than deleting one count digit. To correct a mistyped count, cancel, press f 7 again, and enter the count afresh. A mode-button press also cancels without switching modes; press the desired mode button again afterward.

An empty count or one above the current working precision is rejected when you press =. The prompt ends, the previous display setting remains, and the suspended calculation is unchanged. Leading zeros in a valid count are allowed.

Adjust ALG formatting in the middle of a number

Use ALG, real mode, 600 working digits, and no pending command or shift. Press the unshifted CLEAR, then enter:

2 + 1.20 f 7 6 =

The flag becomes **FIX 6**. The current numeric entry is still 1.20; the expression has not been evaluated. Continue with 3 =. The extra digit extends that operand to 1.203, so the result is 3.203000.

5.5 Reading long numbers

When a number is wider than its display region, drag the digits horizontally. Real RPN and both ALG displays use larger digits and an overflow-only ruler without scrollbars. This also applies to the real RPN full-stack view. Complex RPN offers a scrollbar beneath each component. Moving the digits left reveals later digits; moving them right returns toward the beginning. Scrolling changes only the view. It neither changes the stored value nor rounds it again. A new completed result normally starts at its leading digits.

In the main complex display, a completed result that changes either component returns both rows to their leading digits. This includes a change to the imaginary component alone. The rows remain independently scrollable afterward.

In scientific or engineering notation, the coefficient scrolls while the ×10 and raised exponent stay pinned at the right. Always read that exponent with the visible digits. A portion such as `23456` seen midway through a long coefficient is not a complete number on its own. During exponent entry, the display preserves the sign and digits you are typing, including an unfinished exponent; pinning does not complete it for you.

The **CAL** control in the header appears when a number overflows. It shows or hides a digit ruler without consuming a pending shift or command. If **CANCEL** is also showing, it stays in its fixed position beside the branding. The ruler marks the decimal point with 0 and then counts digits in groups of five outward on either side: 5, 10, 15, and so on. For an integer without a visible decimal point, the zero position is immediately after its last digit. Signs, commas, and exponent digits do not count as coefficient digits. The ruler moves with the coefficient, so its marks stay aligned as you scroll. It is a counting aid, not a statement about how many digits are trustworthy.

Inspect the digits of pi

Use RPN, real mode, fixed depth 5, and 600 working digits, with no pending shift or command. Press:

f **CLEAR** 100 f **7** f **0**

The orange function of **0** supplies π. Under **FIX 100**, its leading digits are `3.1415926535897932384`6…. Drag left to inspect later digits, and use **CAL** to show or hide the five-digit marks. Press f **8** to select **SCI 100**; the coefficient still begins with the same digits, and the pinned exponent is zero. Scrolling and toggling the ruler preserve the value.

Each row of the expanded RPN stack has its own horizontal inspection area. Vertical movement selects deeper registers, while horizontal movement reveals more digits within a row; see section 2.6. In complex mode the two component rows have

separate scrolling regions and pinned exponents. Polar angles also retain their angle-unit suffix. The meaning of those components is covered in chapter 8.

5.6 Display rounding and value rounding

FIX, SCI, and ENG round a presentation of the stored value. They do not replace that value with the displayed decimal. Display rounding uses nearest, ties to even: if a value is exactly halfway between two display choices, the last retained digit is even. For example, exact 0.125 under FIX 2 displays as `0.12`, while exact 0.375 displays as `0.38`. Under FIX 0, 2.5 displays as `2` and 3.5 as `4`.

Those examples use numbers exactly representable in binary. A decimal that merely looks like a tie may be stored just above or below the mathematical halfway point because of input or earlier arithmetic rounding. Judge the actual stored value, not only the visible prefix of its decimal expansion.

The undisplayed fraction still participates

Use RPN, real mode, fixed depth 5, and 600 working digits, with no pending command or shift. Press:

f CLEAR 2 f 7 1 ENTER 8 ÷

The exact quotient is 0.125, displayed as `0.12` under FIX 2. Continue with 8 ×. The result is `1.00`, because the calculation used 0.125 rather than the displayed 0.12.

RND deliberately creates a rounded value. It is the blue function of PMT on the Financial panel and is available in both entry modes. It rounds the real component to the current display count of decimal places, using halfway cases *away from zero*. In RPN it replaces the current X (S0) and saves that operand's unrounded value separately in Last X. The current Y (S1), deeper stack positions, and the saved Last Y operand are unchanged. It does not overwrite a financial or ordinary memory register merely because a value originally came from one.

The count used by RND is a number of decimal places in the value itself, even when SCI or ENG is selected. For example, SCI 2 displays 1234.567 as 1.23×10^3, but RND at that count rounds the value to 1234.57, not 1230. Choose FIX before financial rounding when you want the readout to make the decimal-place convention clear.

In ALG, first finish a compound expression with =, or enter a single number. Press g PMT to round that value and make it the displayed result and ANS. An unfinished compound expression is preserved with the message `Finish the expression first`; RND does not implicitly evaluate it. After a successful rounding,

digits start a new expression and an operator continues from the rounded result. The saved RPN workspace is untouched.

Make the rounding part of the calculation

Use RPN, real mode, fixed depth 5, and 600 working digits, with no pending shift or command. Select the Financial panel as described in section 1.5, then press:

f CLEAR 2 f 7 0.125 ENTER g PMT

Before RND, FIX 2 shows 0.12; after RND it shows 0.13. Continue with 8 ×. The result now displays as 1.04. Explicit rounding has changed the operand used by the multiplication. No financial-register contents are needed for this example.

NINT, the blue function of 9, changes a value by rounding it to the nearest integer, with ties to even. It is available in both entry modes. Thus NINT sends 2.5 to 2, 3.5 to 4, and −2.5 to −2; its integer rounding is independent of the display count. Other integer operations use different rules: INT truncates toward zero, floor rounds downward, and ceiling rounds upward. Their key sequences and domains are in chapter 7; financial rounding is also discussed in chapter 13.

5.7 Existing values and a new working precision

Every stored number retains the precision at which it was created. Changing Working digits does not rewrite the RPN stack, ordinary memories, financial or statistical registers, the previous ALG answer, or numbers already present in an ALG expression. Recalling an older value brings back that value, with its existing precision. Increasing the display count may reveal more of its binary approximation, but cannot supply digits lost when it was created.

A new operation uses the current working precision for its result, even if its operands were created at other precisions. Raising the setting therefore allows more precise subsequent arithmetic, but the new result still depends on the information in those operands. Adding zero to a low-precision approximation at a higher setting does not restore the intended exact number. Lowering the setting likewise leaves old values intact, while new results may retain less detail.

When higher precision matters, select it before entering the original data and performing the calculation. Re-enter approximate decimals from their source, generate constants again, and repeat the necessary operations. Reusing a previously rounded intermediate result is not the same as recomputing that intermediate result at the new precision.

Recover detail by repeating the calculation

Use RPN, real mode, fixed depth 5, and no pending shift or command. Open PREC with g 7; set working digits to 16 and display digits to 6, then press Apply. Press 6 f 7 to select FIX 6, then f 8 to select SCI 6. Clear and calculate:

f CLEAR 1 ENTER 1 EEX +/− 20 + 1 −

This evaluates $(1 + 10^{-20}) - 1$. At 16 working digits, the addition loses the small increment; the final readout is 0.000000×10^{0}.
Open PREC again and change working digits to 40, keeping display digits at 6. Apply. The existing zero remains zero. Repeat the complete clear-and-calculate sequence above. Now the readout is 1.000000×10^{-20}. More working precision retained enough of the small increment for these displayed digits; the result is still a finite-precision approximation, not proof of an exact 10^{-20}.

This example illustrates *cancellation*: the leading digits of nearly equal quantities disappear on subtraction, exposing the accuracy of their much smaller difference. The same issue can arise inside a formula even when its final answer has an ordinary size.

5.8 Choosing useful precision and recognizing limits

The default 600 working digits provide considerable room for ordinary arithmetic. You usually need only a small display count for a price, measurement, or quick comparison. Choose the display for the task: FIX for a familiar decimal scale, SCI for orders of magnitude, or ENG when exponents in groups of three are convenient. Keep working precision comfortably above the answer digits you need, especially when intermediate values differ greatly in size.

For a demanding calculation, repeat it from the original inputs at a higher working precision and compare the digits you intend to use. Agreement is a useful check, but not a proof of correctness: an incorrect formula, an uncertain input, or a numerical limitation can persist at several settings. A more stable mathematical form can be more effective than simply asking for more digits. More precision can also require more time and memory, particularly for long formulas or complex special functions.

Magnitude, domains, and practical bounds

Working precision and exponent range are separate limits. SpicyRPN can represent finite values such as 10^{5000} and 10^{-5000}; the formatter also handles powers such as 10^{100000} without expanding them into an enormous line of integer zeros. A setting

of 600 digits does not mean that values must lie between 10^{-600} and 10^{600}. The underlying exponent range is nevertheless finite. Increasing precision is not a general cure for overflow or underflow.

Other limits belong to particular controls or functions:

- **PREC** accepts 16–10 000 working digits, with the display count from zero through that working count. These are the calculator's exposed setting bounds.
- Real-mode factorial accepts nonnegative integers through 10 000. Permutations and combinations require integers with $0 \leq r \leq n \leq 10\,000$. More working digits do not enlarge those argument ranges.
- For a nonreal argument, complex erf and erfc have an approximate magnitude limit of 100 and may report a failure to converge. Their real-argument path has different numerical behavior; see section 8.9 and chapter C.
- ALG limits a streaming numeric literal to 10 000 characters and an expression to 4 096 tokens and 128 nested groups. These are entry limits, not a promise of answer accuracy.

A mathematically invalid real operation, such as the square root of a negative number in real mode, is not fixed by increasing working precision. Some RPN arithmetic can display ∞, $-\infty$, or `NaN` rather than a finite result; `NaN` means "not a number." ALG rejects nonfinite values during expression evaluation. Other commands issue specific domain or convergence messages. Read the status area and check the operands and selected mode before continuing.

Complex arithmetic and elementary functions use MPC, which rounds each component of an individual primitive at the precision requested by the calculator. A composed calculation, angle conversion, or custom special-function approximation does not inherit a blanket guarantee of correct rounding for its complete result. Such algorithms can still lose digits through cancellation or fail to converge; a convergence message does not mean that the mathematical function is undefined. Chapters C and 8 discusses complex conventions and numerical interpretation.

When an answer surprises you, first check the displayed exponent, notation, entry mode, angle unit, and complex setting. Next check the operands and any reused intermediate values. Only then decide whether to repeat the calculation at a higher working precision. The diagnosis and recovery procedures in chapter 14 complement these numerical checks.

Chapter 6

Storing and Reusing Values

Memory keeps a value available while the stack or expression moves on to other work. Store a conversion factor, a measured quantity, or an intermediate result once, then recall it wherever it is needed. The stored value remains until you replace it or clear its bank; recalling it does not use it up.

SpicyRPN has several memory banks. Ordinary registers serve everyday calculations in both entry modes. Associative memory identifies values by numeric keys in both modes. Statistics and finance use dedicated banks for their own inputs and results. Understanding these distinctions also makes it easier to clear exactly the data you intend to replace.

6.1 The memory banks and their indicators

Table 6.1 summarizes the banks and the display flags that announce their contents. A flag means that the bank contains at least one stored entry, including an explicitly stored zero. It does not identify which register is occupied, show a count of entries, or certify that the data is appropriate for your next calculation.

Table 6.1: Stored data and its occupancy flags.

Bank	Flag	Purpose and access
Ordinary	**MEM**	Ten registers, R0–R9, for values you choose. STO and RCL work in RPN and ALG.
Associative	**ASSOC**	Values stored under numeric keys. Access through the orange functions of STO and RCL in both modes.
Statistics	**STAT**	Six accumulators for observations, sums, and products. RPN keys or ALG STAT worksheets.
Financial	**FIN**	The five registers n, i, PV, PMT, and FV. Shared by both entry modes.
Cash flows	**CF**	An initial flow and subsequent equally spaced flows, entered and used in both modes.

These banks are separate from the working stack. In RPN, X is stack position S0, and Y is S1. R0 is ordinary memory register 0, not another name for S0. Similarly, the statistical sample count and the financial period count are distinct values even though both are conventionally written n.

The memory flags describe shared banks in either entry mode. A **STAT**, **FIN**, or **ASSOC** flag describes the same stored data whether you use RPN commands or ALG worksheets. The banks and their flags survive closing and reopening the application; see section 6.8.

6.2 Ordinary storage and recall

Choose any digit from 0 through 9 as a register number. To store, press **STO**, then that digit. To recall, press **RCL**, then the digit. For example, **STO** **3** selects R3. The digit is a destination, not a new number being entered. There is no second digit in an ordinary register address.

STO and **RCL** are the unshifted memory keys beside the statistics controls in RPN and beside the expression controls in ALG. Pressing either key arms a command: the **STO** or **RCL** flag and status prompt show that a target is pending. The operation finishes when you choose its register. Any number still being typed is committed at that point.

An unshifted backspace cancels a pending ordinary **STO** or **RCL** without deleting a digit. **CLx** or **CLEAR** also cancels that pending command rather than clearing an operand. The header's **CANCEL** dismisses the entire pending selection, including an arithmetic STO modifier and any active shift, while preserving your unfinished entry. With only a shift active, pressing it again also cancels it. See section 6.10 for MCLR selectors.

In RPN

The current X (S0) and Y (S1) hold working operands. The separate Last X and Last Y hold previous operands saved by commands; they are not stack positions.

STO copies X into the selected register. It replaces any previous value there and leaves the stack and both saved operands, Last X and Last Y, unchanged. You can therefore store a result and continue calculating with it.

Ordinary **RCL** lifts the recalled value into X. The previous X moves to Y, with deeper values moving one position farther away. It also saves that previous X in Last X; Last Y is unchanged. If you were typing a number, recall first commits it, then moves it to Y as the recalled value enters X. The normal fixed-stack limit or unlimited-stack rules apply, as described in chapter 2.

Put a result aside, then use it again

Use RPN, real mode, a fixed stack of 5, and FIX 4, with no active shift or pending command. This example replaces R2; other memories may contain anything. Clear the stack with f CLEAR, the orange function of $1/x$, then enter:

12 ENTER 3 × STO 2

The display remains 36.0000, and R2 now contains 36. Clear the stack again and calculate with that value:

f CLEAR 9 RCL 2 +

Recall commits 9 and puts 36 in X, with 9 in Y. Addition gives 45.0000. R2 still contains 36.

In ALG

STO stores the current operand or completed result. It commits an active numeric entry but does not evaluate the whole unfinished expression. With $2 + 3$ entered and no = yet pressed, STO 2 stores 3. To store the sum, press = first, then STO 2.

RCL inserts the recalled value where an operand is expected. If the expression already ends in an operand, it replaces that last operand, including a completed parenthesized group or function result. After a completed calculation, recall starts a new expression containing the recalled value. It does not append a second operand by implicit multiplication. Neither ordinary storage nor recall changes ANS; the next successful = updates that answer.

Store an operand or store the expression's result

Use ALG, real mode, and FIX 4, with no active shift or pending command. The main CLEAR key clears the expression; this example deliberately replaces R2.

CLEAR 2 + 3 STO 2

The unfinished expression is still $2 + 3$, the current value is 3.0000, and R2 contains 3. Continue with:

= STO 2

Now the display and R2 both contain 5. Try each row below independently. Each starts by clearing the expression, and all use this same stored 5.

Key sequence	Result
CLEAR 10 + RCL 2 =	15.0000
CLEAR 10 + 99 RCL 2 =	15.0000
CLEAR 12 = RCL 2 =	5.0000

In the first row, recall supplies the missing operand. In the second, it replaces 99. In the third, it begins a new expression after 12.

6.3 Arithmetic storage

You can change a stored value without recalling it, performing arithmetic on the stack or in the expression, and storing it again. Press STO, then one of the four arithmetic keys, then the register digit. The operator modifies the pending storage command; it does not perform the ordinary on-screen arithmetic operation.

Let M be the value already in the destination and v the current operand: X in RPN, or the current ALG operand or completed result. The order is always M followed by the operator and v, as shown in table 6.2.

Table 6.2: Arithmetic storage in ordinary register R3.

Sequence	New R3	Pending flag
STO + 3	$M + v$	STO+
STO − 3	$M - v$	STO−
STO × 3	$M \times v$	STO×
STO ÷ 3	M/v	STO÷

Subtraction removes the current operand from memory; division divides memory by the current operand. Both operations leave the current operand in place. In RPN they preserve all stack positions and both Last registers after committing any entry. In ALG they leave the expression unevaluated and use the same current-operand rule as ordinary storage. To add an expression's total to memory, finish it with = before pressing STO +.

An unset register supplies zero as M. Thus adding 5 to an unset register stores 5, subtracting 5 stores −5, and multiplying an unset register by 5 stores zero. Each successful operation creates a stored entry. Division by a zero current operand reports an error and leaves the destination unchanged.

Reduce and divide a stored amount

Use RPN, real mode, a fixed stack of 5, and FIX 4, with no active shift or pending command. The following sequence initializes R1, so its previous contents do not matter:

f CLEAR 20 STO 1 5 STO − 1

R1 now contains $20 - 5 = 15$, while the display is `5.0000`. Continue:

3 STO ÷ 1 RCL 1

The arithmetic store changes R1 to $15/3 = 5$ and leaves 3 in X. Recall then displays `5.0000`, with 3 in Y.

6.4 Reusing a constant

A stored constant lets several calculations use exactly the same value. Choose a register, store the factor, and recall it for each calculation. A result's displayed decimal places do not limit what STO saves: storage and recall use the underlying digits, not the rounded display. Both complex components are copied when CMPLX is on; with CMPLX off, only the real component is copied, as explained in section 6.5. Display changes do not round the contents of a memory register. See chapter 5 for the separate effects of working and display precision.

Apply one scale factor to three quantities

Use RPN, real mode, a fixed stack of 5, and FIX 4, with no active shift or pending command. Set R0 to the scale factor 2.5:

f CLEAR 2.5 STO 0

Then follow each row. Clearing the stack between quantities keeps the examples independent and preserves the stored factor.

Key sequence	Scaled quantity
f CLEAR 12 RCL 0 ×	30.0000
f CLEAR 18 RCL 0 ×	45.0000
f CLEAR 7.2 RCL 0 ×	18.0000

There is no need for ENTER between a quantity and RCL: recall finishes its entry and lifts the factor into X.
To use the same factor in ALG, select ALG, retaining real mode and FIX 4. With no pending command, enter:

CLEAR 12 × RCL 0 =

> The result is again `30.0000`. Both modes use the same R0.

If a constant needs higher precision, recompute it and replace its stored value; see section 5.7.

6.5 Complex values in memory

The complex setting determines the value written by a store or recall. With **COMPLEX** on, both components of the value are copied. With it off, only the real component is copied and the destination's imaginary component is zero. This applies to ordinary and associative memory, financial-register and cash-flow assignments, and the working register or ALG operand receiving a recall. Arithmetic storage also uses the current complex setting. Statistical calculations and financial formulas use real components even with CMPLX on; their newly computed results and statistical totals are real.

Changing the setting does not rewrite existing memories. A complex value stored while CMPLX is on keeps both components when CMPLX is turned off. Recalling it in real mode creates a real working copy; the source memory remains unchanged. Turn CMPLX on and recall it again to obtain the original pair. In contrast, storing into that memory while CMPLX is off replaces its old pair with a real value, even if the source operand still has a hidden imaginary component.

Newly saved Last X and Last Y values follow the same rule. Merely moving existing stack values by lifting, dropping, swapping, or rolling preserves their components; an idle **ENTER** creates a new copy in X, so that copy is real when CMPLX is off. See section 8.6 for a complete store/recall example.

6.6 Unset registers and stored zero

Recalling an unset ordinary register returns zero without creating an entry in that register. The recall therefore does not light **MEM**. By contrast, entering zero and pressing **STO** followed by a digit stores a real entry whose value happens to be zero; **MEM** is then present.

The same distinction underlies all occupancy flags. An unmatched associative key returns zero without creating a pair. Statistical accumulators can contain zeros and still occupy their bank. Removing observations until the sample count is zero does not empty that bank. A zero financial input remains stored, and a cash-flow list containing zero-valued flows still lights **CF**.

Consequently, a zero recall alone cannot tell you whether a particular register was never used, was cleared, or explicitly contains zero. Nor can **MEM** identify which

of the ten ordinary registers is occupied. Use deliberate initialization when the distinction matters. For example, start an ordinary running total by storing zero in its chosen register, then use STO + for each amount.

Overwriting one register with zero preserves the other registers but does not remove occupancy. The grouped MCLR operation removes entries from its selected bank; it is not an individual-register selector. A complete demonstration appears in section 6.10.

6.7 Associative memory: values under numeric keys

Associative memory lets a number identify another number. For example, you can use an item number as the key for its measured value. These keys are numeric values, not text names and not ordinary register numbers. Key 123 in associative memory has no connection to the ordinary digits used after STO or RCL.

In RPN, the associative commands use stack operands. Orange f STO selects the legend STO_{xy}; orange f RCL selects RCL_{x}. Unlike ordinary storage and recall, these commands act immediately on operands already on the stack. There is no following register-selection digit.

Keys match by numerical equality, not by the spelling of their entry. For example, exact values entered as 1 and 1.0 identify the same key; storing again replaces the earlier value. A rounded display is not an equality test: two computed keys that both display as 1.0000 may have different hidden digits and fail to match. Exact, easily repeated keys such as integers are convenient. The key's real component must not be NaN; its imaginary component is ignored.

To store a pair, put the *value* in Y and its *key* in X. In the usual entry sequence, type the value, press ENTER, then type the key and press f STO. Storage saves the old X and Y in Last X and Last Y, then drops the key from X. The stored value moves down from Y into X. A fixed stack repeats its deepest value; an unlimited stack shrinks by one and must have both operands.

To retrieve the value, put its key in X and press f RCL. Recall replaces the key with the stored value, without lifting the stack. It saves the key in Last X and preserves Last Y. A key with no matching entry is replaced by zero.

Look up a value by item number

Use RPN, real mode, a fixed stack of 5, and FIX 4, with no active shift or pending command. This example stores 42 under key 123, replacing any existing value with that matching key. Other associative entries may remain.

f CLEAR 42 ENTER 123 f STO

The display returns to 42.0000; the key has been dropped. **ASSOC** is present, Last X contains 123, and Last Y contains 42.
To add the stored value to 9, enter:

f CLEAR 9 ENTER 123 f RCL +

Typing 123 places 9 in Y. Associative recall replaces 123 with 42 while leaving 9 in Y; addition gives 51.0000. The stored pair remains available.

One real key in either mode

Associative storage and recall identify an entry by the numeric key's real component, whether CMPLX is on or off. Thus $3 + 4i$, $3 - 8i$, and 3 all address the same entry. Storing with any of those keys replaces the value previously stored with that real key. The key itself is stored as a real number; an imaginary component does not distinguish entries.
The stored *value* is separate from the key and can be complex when CMPLX is on. RPN takes the key from X (S0) and the value from Y (S1); ALG uses its labeled Key and Value worksheet fields, without using the RPN stack. With CMPLX off, a new store copies only the value's real component. A real-mode recall also returns only the real component, without changing the stored pair; a later complex-mode recall returns both. See sections 6.5 and 8.6.

Associative memory in ALG

Worksheet fields and recalled outputs must be finite. If an entry stored from RPN contains NaN or infinity, associative recall in ALG reports an error rather than making that nonfinite value an output.

Orange STO, labeled STO_{xy}, opens a worksheet with Key and Value. Enter and accept both fields with ENTER, then choose STORE. This writes the pair only after both inputs validate. Orange RCL, labeled RCL_x, opens a worksheet with Key; accept it, then choose RECALL. The output is the stored value, or zero if there is no match. Choose Use value to insert it into the suspended expression.

With complex mode on, the Value field can contain a complex expression. The real-key and component-copying rules in the note above apply here. The primary

output of successful STORE or RECALL updates ANS; accepting a field updates ANS too. No RPN stack value or Last register participates.

For example, accept Key 123 and Value 42, then STORE. To use it in $9+42$, return to the calculator, start 9+, open associative recall, accept Key 123, choose RECALL, then **Use value** and **=**. The answer is 51. CANCEL closes the worksheet without inserting an output; completed storage remains. The shared worksheet rules are in section 4.12.

6.8 Memories, working areas, and saved operands

RPN and ALG keep independent working areas but share the memory banks. Storing in R3 in ALG replaces the value that RPN will later recall from R3. Mode switching preserves each working area without transferring a value; see section 4.10.

The effects of the main RPN memory commands are collected in table 6.3. Here X means the current S0 and Y the current S1; the Last registers hold separately saved previous operands. All effects refer to the committed operands: if a number is being entered, it is finished before the command acts. Ordinary and arithmetic storage preserve operands; recall and associative storage have the movements shown. When CMPLX is off, each new memory value, recalled working value, and saved Last operand is real, as described in section 6.5.

Table 6.3: RPN memory effects after any active number is committed.

Command	Stack effect	Saved operands
Ordinary **STO**, including arithmetic storage	No movement; X stays in place.	Both Last registers preserved.
Ordinary **RCL**	Lift recalled value into X.	Old X becomes Last X; Last Y preserved.
Associative **f** **STO**	Store Y under key X, then drop X.	Save the key in Last X, the value in Last Y.
Associative **f** **RCL**	Replace key X with its value; no lift.	Save the key in Last X; Last Y preserved.
Financial **STO**, including arithmetic storage	No movement; X stays in place.	Both Last registers preserved.
Financial-register **RCL**	Lift recalled value into X.	Both Last registers preserved.

ALG memory operations do not touch the suspended RPN stack or its Last registers. They act on the ALG operand or expression as described in section 6.2. Clearing

an ordinary ALG expression, or using **CLx** or **CLEAR** on the RPN working area, leaves all these memory banks intact.

Stored values, the independent working areas, and pending commands are saved automatically. Relaunching does not empty a register or cancel a partly entered **STO** sequence. Read the command flags before resuming. Saved operands are also not a backup of a memory bank: **LAST X** and **LAST Y** cannot undo a store or reconstruct cleared memories. See chapters 2 and 14.

6.9 The statistical and financial banks

Statistics

The statistical bank holds six accumulators: the sample count n, $\sum x$, $\sum x^2$, $\sum y$, $\sum y^2$, and $\sum xy$. Adding or removing observations changes these totals; it does not maintain a retrievable list of the original observations. The statistics commands in both modes use this bank without borrowing ordinary registers for the running accumulation.

RCL_{Σ}, the blue function of **L.R.**, copies those six totals into ordinary registers R4–R9, in the order shown below. This *overwrites* any values already stored there. The statistics bank itself is unchanged, and the command does not lift six values onto the stack.

Ordinary register	R4	R5	R6	R7	R8	R9
Copied total	n	$\sum x$	$\sum x^2$	$\sum y$	$\sum y^2$	$\sum xy$

You can then inspect a total with ordinary **RCL** and its digit, including after switching to ALG. Even when the statistical bank is empty, copying its totals stores six zeros and lights **MEM** without lighting **STAT**. Subsequent changes to the observations do not update the copied values; repeat the copy when you need fresh totals. Chapter 10 covers sample entry, correction, and the statistical results.

Financial registers and cash flows

The five financial registers hold the period count n, the interest percentage per period i, present value PV, periodic payment PMT, and future value FV. In RPN, their main keys on the Financial panel are also targets for explicit storage and recall. Use **STO** **PV** to store X in PV, for example, or **RCL** **PV** to lift its value onto the stack. Arithmetic storage works with these targets too. Explicit **STO** chooses storage regardless of whether the next bare financial key would otherwise solve for a variable.

In RPN, bare financial keys either store or solve, according to the preceding action. After an explicit financial store, the next bare financial key solves; an ordinary recall permits the next bare financial key to store. For the complete store/solve rules, input signs, rate periods, and payment timing, follow chapter 11. A solve uses all four other financial registers, including stored zeros. An old payment or future value can therefore affect a new problem until you replace it or clear it.

The cash-flow bank is a separate sequence used for NPV and IRR. Its commands also use the financial n register as an index. In particular, CF_0 starts a new cash-flow list and stores $n = 0$, so it lights **FIN** as well as **CF**, even when the initial flow is zero. Indexed storage and recall use CF_j as their target and have different operand requirements from financial-register storage; use the procedures in chapter 12.

For separate financial clears in either mode, orange f **FV** selects CL_{fin} and removes the five financial registers while leaving cash flows. Blue g **FV** selects CL_{cf} and removes the cash-flow list, but also stores zero in n; **FIN** therefore remains present. The financial **MCLR** group clears both banks together. In ALG, bare register keys always store the current operand; CPT followed by a register explicitly solves it. Financial recall inserts like ordinary RCL. The cash-flow keys open labeled worksheets; STO or RCL followed by CFj opens indexed storage or recall. See sections 11.12 and 12.9.

6.10 Clearing the intended group with MCLR

MCLR is the blue function of **STO**. With no existing shift or pending command, press g **STO**. The **MCLR** flag appears and the prompt asks you to choose a group. The next key selects the group to clear, as summarized in table 6.4. Its ordinary function is not performed. Do not insert a shift between **MCLR** and a desired selector: it cancels **MCLR** instead of choosing an alternate legend.

The digit 0 in the first row is a convenient selector for the entire ordinary-and-associative group. It does not mean "clear only R0." The entry/arithmetic group includes the digits, decimal point, **EEX**, sign change, operators, and **ENTER** or **=**. In ALG, every Expression key—the parentheses, **FUNC**, **CPT**, and **CLEAR**—and the entry key $1/x$ also select this memory group. In RPN, $1/x$ belongs to the stack group instead.

Keys outside the clearing groups, such as a Scientific-panel key or **STO**/**RCL** themselves or ALG's **ANS**, also cancel **MCLR** without performing their normal function. The group is determined before ordinary key routing.

You may swipe to the Financial panel while **MCLR** is pending; paging does not consume the command. In either mode, pressing **n** then clears both financial banks without performing its ordinary action. For the MCLR statistics or stack selector, use RPN. ALG also provides Clear statistics through its STAT worksheets.

Table 6.4: One-key selection after g STO has armed MCLR.

Press next	Mode	What is cleared
0	RPN or ALG	All ordinary registers *and* all associative entries. Entry/arithmetic keys except backspace select this group.
Σ+	RPN	All six statistical accumulators. The mean and L.R. keys select the same group.
n	RPN or ALG	All five financial registers *and* the entire cash-flow list. Any Financial-panel key selects this group.
R↓	RPN	The whole RPN stack and unfinished RPN entry. $x \leftrightarrow y$ and $1/x$ select the same group.
CANCEL or backspace	RPN or ALG	Nothing: cancel MCLR. Either shift or physical Escape also cancels. A shift press is consumed and leaves no shift active.

Clearing a memory bank removes its entries and corresponding occupancy flags while preserving the active number or expression, the working stack, and both Last registers. It does not even finish an unfinished numeric entry. Clearing the *stack* group instead discards the RPN entry and zeros the stack, preserving both Last registers and all memory banks. Neither operation resets precision, display format, angle units, complex mode, or payment timing. MCLR is a group clear, not a factory reset.

Distinguish an empty bank from a stored zero

This example intentionally removes *all ordinary and associative memories*. Use it when you are ready to discard those entries. Use RPN, real mode, a fixed stack of 5, and FIX 4, with no active shift or pending command.

f CLEAR 84 STO 9

R9 contains 84 and **MEM** is present. Now press:

g STO 0

All ordinary and associative entries have been removed. Their flags are absent, but X still displays `84.0000`. Press RCL 9. The display becomes `0.0000`, 84 moves to Y, and **MEM** stays absent because recall has not created an entry. Finally press STO 9. This stores the recalled zero in R9. The display is still `0.0000`, but **MEM** is now present. The display value is the same; the bank's occupancy is different. Statistical, financial, and cash-flow data are unchanged throughout.

Chapter 7

Scientific Functions

The scientific keys extend ordinary arithmetic to powers, logarithms, angles, counting problems, and special functions. Most need one operand; a few need two, whose order matters. The same mathematical keys are available in RPN and ALG, but the entry sequence follows the working rules of the selected mode.

This chapter concentrates on real-number calculations. Its domains and examples assume that **COMPLEX** is off. Complex mode changes the meaning and domain of several functions, including powers, roots, and factorial; chapter 8 explains those extensions and their limits. Working precision and display rounding remain the separate choices explained in chapter 5.

7.1 Finding and applying a scientific function

Show the Scientific panel by swiping right across the top two keyboard rows in portrait. In landscape, the scientific keys are already in the left bank. Table 7.1 maps every main and shifted function in those two rows. Press f for the orange column or g for the blue column, followed by the key in the first column. For example, f LN means orange shift followed by the e^x key carrying the LN legend.

Table 7.1: The Scientific panel, in keyboard order.

Main key	Orange function	Blue function
SIN	DEG	SIN^{-1}
COS	RAD	COS^{-1}
TAN	GRAD	TAN^{-1}
HYP	HYP^{-1}	atan2
Γ	$\ln\Gamma$	nPr
$\sqrt{x}$	x^2	erf
e^x	LN	erfc
10^x	LOG	LOG_2
y^x	$\sqrt[x]{y}$	$\log_y x$
$x!$	$\ln x!$	nCr

Other scientific operations use alternate legends on the fixed keys. Their physical locations are collected in table 7.2. Settings such as FIX, FSE, and PREC are covered in chapter 5; complex controls and CONV are covered in chapters 8 and 9.

Table 7.2: Scientific functions outside the sliding panel.

Physical key	Scientific alternate functions
ENTER (RPN)	Orange INT; blue FRAC.
$1/x$ (ALG)	Orange INT; blue FRAC.
9	Orange DRG, to cycle angle units; blue NINT.
4	Blue RAN#.
5	Blue ceiling, $\lceil x \rceil$.
6	Blue floor, $\lfloor x \rfloor$.
×	Orange ABS; blue REM.
÷	Blue MOD.
1, 2, 3	Blue %, Δ%, and T%, respectively.
0	Orange π.
.	Orange e, Euler's number.
←	Orange $\ln 2$.

A reproducible starting point

Unless an example specifies otherwise, use real mode, 600 working digits, and FIX 4. RPN examples use a fixed stack of 5. Set these through the procedures in sections 2.2, 5.2 and 5.4. Set any required angle unit before entering functions. Check that there is no active shift, pending command, or open tool before starting a sequence.

In RPN, f CLEAR is the orange function of $1/x$; in ALG, CLEAR is an unshifted editing key. The worked examples below include the appropriate clear. These clears leave memory banks intact, and no example depends on their earlier contents.

One operand, two operands, or a new value

In RPN, a unary function finishes any active entry and replaces X, the current value displayed at S0. A binary function uses Y (displayed at S1) first and X second, normally dropping the stack afterward. The three percentage functions instead keep Y and every deeper position. Constants and RAN# need no operand: they lift a new value into X. Last X and Last Y are saved operands from earlier operations, not the current S0 and S1 values. See sections 2.5 and 2.9 for their capture and recall rules.

In ALG, a unary function can precede its argument or follow the operand to which it applies. For example, SIN 30) = and 30 SIN = are two ways to calculate the sine of 30 in the selected angle unit. A binary key normally goes between its two operands. Alternatively, FUNC followed by the function opens a call with two arguments; separate them with g FUNC and finish with). The first argument

corresponds to RPN's Y, the second to X; ALG has no current S0/S1 stack positions. Mathematical symbols such as x and y describe the arguments of a formula, not extra registers. Chapter 4 explains grouping and precedence.

7.2 Arithmetic, reciprocal, powers, and roots

Addition, subtraction, multiplication, and division use their ordinary keys. In RPN their order is $Y + X$, $Y - X$, YX, and Y/X. The $1/x$ key replaces a number with its reciprocal. Division and reciprocal require a nonzero divisor for a finite answer. The sign-change key +/− negates an operand; while a number is being entered, it edits the sign of the coefficient or the active exponent instead.

Square root is the main $\sqrt{x}$ key, and square is its orange function. A real square root requires $X \geq 0$; the result is the nonnegative root. The square of a negative number is positive, so squaring followed by square root returns its absolute value, not its original sign.

For y^x, put the base in Y and exponent in X, or enter base, power key, exponent in ALG. A positive base accepts any finite real exponent, subject to numerical range. A negative base requires an integer exponent for a real result from this key. A fractional exponent such as a calculated 1/3 does not request the real cube root of a negative number. Use the indexed root key for that purpose.

The orange $\sqrt[x]{y}$ function takes the radicand first and the root index second. For an integer index, real mode accepts nonzero integers from −1 000 000 through 1 000 000. An odd index permits a negative radicand; an even index requires a nonnegative one. A negative index returns the reciprocal of the corresponding positive-index root, so the radicand must be nonzero for a finite result. Index zero is rejected.

A noninteger index is also accepted for a nonnegative radicand and is evaluated as $Y^{1/X}$. It cannot obtain a real root of a negative radicand. Zero raised to a negative power, or used with a negative root index, has no finite result. These restrictions describe real mode; complex principal roots can differ even when a real root exists.

Choose the real cube root and a reciprocal root

Use the chapter's RPN starting state. Enter:

f CLEAR 125 +/− ENTER 3 f $\sqrt[x]{y}$

The result is `-5.0000`, since $(-5)^3 = -125$. For an independent calculation, enter:

f CLEAR 16 ENTER 2 +/− f $\sqrt[x]{y}$

The index is −2; the result is $1/\sqrt{16} = 0.25$, displayed as `0.2500`.

Combine a square and square root in ALG

Use the chapter's ALG starting state. Calculate $\sqrt{3^2 + 4^2}$:

CLEAR (3 f x^2 + 4 f x^2) $\sqrt{x}$ =

The display is `5.0000`. The closing parenthesis makes the entire sum, 25, the operand of square root. Without that group, a postfix square root would apply only to the preceding operand.

7.3 Exponential and logarithmic functions

e^x raises Euler's number to the operand, and 10^x raises 10 to it. They accept positive, zero, and negative real exponents. They are functions of an existing number; EEX, by contrast, introduces a power-of-ten exponent while *typing* a number. For example, entering 2 EEX 3 means 2000, whereas applying 10^x to 3 produces 1000.

The three single-operand logarithms differ only in base: f LN gives $\ln X$, f LOG gives $\log_{10} X$, and g LOG_2 gives $\log_2 X$. Their finite real domain is $X > 0$. LN and e^x are inverse functions, as are LOG and 10^x, within the precision and range of the calculation.

The blue function of y^x calculates a logarithm to an arbitrary base:

$$\log_Y X = \frac{\ln X}{\ln Y}.$$

Enter the *base first*, then the argument. The base must be positive and different from 1; the argument must be positive. A base between zero and one is valid and reverses the usual sign pattern: $\log_{1/2} 8 = -3$, because $(1/2)^{-3} = 8$. A base extremely close to 1 makes this quotient sensitive to small errors in the base.

Find a power of two and use a fractional base

Use the chapter's RPN starting state. For the binary logarithm:

f CLEAR 32 g LOG_2

The display is `5.0000`. For the arbitrary-base calculation, start again:

f CLEAR .5 ENTER 8 g $\log_y x$

The display is `-3.0000`. The first operand, 0.5, is the base; it is not the number whose logarithm is requested.

Enter a logarithm call in ALG

Use the chapter's ALG starting state. Press:

CLEAR FUNC g $\log_y x$ 3 g FUNC 81) =

The blue FUNC supplies the argument comma. The expression represents $\log_3 81$ and displays `4.0000`. The logarithm key is the blue function of y^x.

7.4 Angle units and trigonometric functions

The angle setting tells SIN, COS, and TAN how to interpret their operand, and tells the inverse functions which unit to return. Select a unit directly with the orange functions of SIN, COS, and TAN, or cycle it with f 9, the DRG function. The cycle is DEG, RAD, GRAD, then DEG again.

Table 7.3: Angle units and direct selection.

Flag	Select with	Full turn	Right angle
DEG	f SIN	360	90
RAD	f COS	2π	$\pi/2$
GRAD	f TAN	400	100

Changing the angle setting does not convert a number already entered or stored. If X contains 30, selecting RAD leaves it as 30 and a following SIN calculates sin(30 radians). Use the angle family in CONV when you want to convert the numerical value itself; see chapter 9.

The setting is shared between modes and can be changed without finishing a numeric entry. In ALG, each trigonometric function retains the unit selected when its function key was pressed. Changing the flag afterward does not change functions already in the expression. Choose the desired unit before inserting the function, and check its unit-bearing expression label if the setting has since changed.

SIN and COS have finite real values for all finite real angles. TAN has poles at odd multiples of a right angle, such as 90° and 270°. An angle close to a pole can produce a very large finite value. More displayed digits do not make an uncertain angle near a pole reliable.

Calculate the same sine in degrees and radians

Use the chapter's RPN starting state. Select degrees in the sequence:

f CLEAR f DEG 30 SIN

The display is `0.5000`. Now clear and use radians:

f CLEAR f RAD f π 6 ÷ SIN

The orange π function is on 0. The sequence calculates $\sin(\pi/6)$, again displaying `0.5000`. The constant lifts directly onto the stack; no ENTER is needed before the following new operand 6.

The blue SIN, COS, and TAN functions are inverse sine, inverse cosine, and inverse tangent. Their superscript -1 means *inverse function*, not reciprocal. To obtain $1/\sin x$, apply SIN and then $1/x$ instead.

Table 7.4: Finite real domains and principal inverse-trigonometric ranges. The ranges are shown in degrees; RAD and GRAD give the equivalent angles.

Function	Input	Result range
g SIN	$-1 \le X \le 1$	$[-90°, 90°]$
g COS	$-1 \le X \le 1$	$[0°, 180°]$
g TAN	Any finite real X	$(-90°, 90°)$

An inverse function returns one principal angle rather than every angle with the same sine, cosine, or tangent. For example, both 30° and 150° have sine 0.5, but inverse sine returns 30°.

A direction from two coordinates

atan2, the blue function of HYP, uses both coordinates to preserve the quadrant. Its order is atan2(y, x): enter the vertical coordinate first and the horizontal coordinate second. In RPN those occupy Y and X. The result is an angle from the positive horizontal axis, in the selected unit, between $-180°$ and 180° or the equivalent in radians or gradians. A positive vertical coordinate chooses the upper half-plane; a negative one chooses the lower. Unlike computing $\tan^{-1}(y/x)$, atan2 retains the signs of both coordinates and accepts a zero horizontal coordinate. The origin has no geometrical direction; do not interpret the numerical convention returned for two zeros as a measured angle.

Keep the quadrant of a direction

Use the chapter's RPN starting state. For the point $(x, y) = (-1, 1)$, press:

f CLEAR f DEG 1 ENTER 1 +/− g atan2

The result is `135.0000` degrees. Dividing the coordinates first would give -1, whose ordinary inverse tangent is $-45°$; that ratio alone cannot identify the quadrant.

7.5 Hyperbolic functions and their inverses

HYP is a selection prefix. Press it, then unshifted SIN, COS, or TAN to choose hyperbolic sine, cosine, or tangent. For an inverse, press f HYP, then unshifted SIN, COS, or TAN. The **HYP** or **INVHYP** indicator remains until the selection is completed. The prefix is used once; the next ordinary SIN is trigonometric again.

Do not insert another shift between HYP and the trigonometric key: that shift cancels the hyperbolic selection. An unshifted backspace also cancels the pending selection without editing the operand. In ALG, use the HYP sequence directly, without putting FUNC before it. The selected unary function opens a call if an argument is needed or acts on the preceding operand.

Hyperbolic functions use numerical arguments, not angles. DEG, RAD, and GRAD have no effect on them. Their definitions include

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2}, \qquad \tanh x = \frac{\sinh x}{\cosh x}.$$

All three accept finite real inputs, subject to numerical range. Inverse hyperbolic sine accepts every finite real input; inverse hyperbolic cosine requires $X \geq 1$ and returns a nonnegative value; inverse hyperbolic tangent requires $-1 < X < 1$ for a finite real result. Its endpoints are singular.

Apply and reverse a hyperbolic cosine in ALG

Use the chapter's ALG starting state. The angle setting may be any unit. Press:

CLEAR HYP COS 1) =

The result, $\cosh 1$, displays as `1.5431`. Continue with:

f HYP COS =

The inverse hyperbolic cosine returns `1.0000`. It uses the retained value of $\cosh 1$, not the four-place readout 1.5431. The final equals press completes the new ALG answer.

7.6 Absolute value, integer parts, and rounding

ABS, the orange function of multiplication, returns the nonnegative magnitude of a real number. The integer functions have different rules for choosing an integer; their differences matter especially for negative inputs. They change the operand itself, independently of FIX, SCI, or ENG.

Table 7.5: Integer and fractional operations on a real operand.

Function and location	Rule
INT: orange ENTER (RPN), orange $1/x$ (ALG)	Truncate toward zero, discarding the fractional part.
FRAC: blue ENTER (RPN), blue $1/x$ (ALG)	Return $X - \text{trunc}(X)$; a nonzero fraction has the input's sign.
NINT: blue 9	Nearest integer; an exact tie goes to the even integer.
$\lceil x \rceil$: blue 5	Ceiling: the least integer greater than or equal to X.
$\lfloor x \rfloor$: blue 6	Floor: the greatest integer less than or equal to X.

For -2.75, INT gives -2, FRAC gives -0.75, NINT and floor give -3, and ceiling gives -2. Thus INT is not the same as floor, and FRAC is not always positive. NINT's tie rule sends 2.5 to 2, 3.5 to 4, -2.5 to -2, and -3.5 to -4. These exact halves are representable in binary; a computed value merely displayed as a half may lie on one side of the tie internally.

Separate a signed number into two parts

Use the chapter's RPN starting state. Try these independently:

Complete sequence	Display
f CLEAR 2.75 +/− f ENTER	-2.0000
f CLEAR 2.75 +/− g ENTER	-0.7500

The first row selects INT and the second selects FRAC. The two mathematical parts sum to the original -2.75. In ALG, use f $1/x$ for integer part, or g $1/x$ for FRAC; then press the separate = to finish the expression.

Financial RND is a separate function in both modes on the blue PMT legend. It rounds to the selected number of decimal places, with ties away from zero. It is neither NINT nor a display-only change; in ALG it requires a completed expression or a single entered number and updates ANS. See section 5.6.

7.7 Two different remainder conventions

Both remainder keys divide the first operand by the second and return what is left after subtracting an integer multiple of the divisor. Their difference is the rule for that integer. With dividend Y and nonzero divisor X,

$$r = Y - qX.$$

MOD, the blue division function, chooses $q = \text{trunc}(Y/X)$, toward zero. Its nonzero result has the dividend's sign and magnitude less than the divisor's magnitude. This is a *truncating remainder*; despite the MOD legend, it is not a rule that always returns a nonnegative residue.

REM, the blue multiplication function, chooses the nearest integer to Y/X, with exact quotient ties going to the even integer. This is the IEEE remainder convention. Its magnitude is at most half the divisor's magnitude, and its sign can differ from the dividend's sign. The quotient selection uses the mathematical quotient for the remainder operation, not a displayed rounded quotient.

Table 7.6: How the remainder conventions differ. All divisors here are 4.

Dividend	MOD quotient	MOD result	REM quotient	REM result
7	1	3	2	−1
−7	−1	−3	−2	1
6	1	2	2	−2
10	2	2	2	2

The last two rows illustrate REM ties: 1.5 and 2.5 both choose the even quotient 2. Both functions also accept noninteger operands; these are real-number remainder operations, not integer-only keys. A zero divisor has no finite remainder.

Compare MOD and REM in both modes

Use the chapter's RPN starting state and try:

f **CLEAR** 7 **ENTER** 4 g ÷

MOD gives `3.0000`. Repeat from CLEAR, replacing the last shifted division key with g × ; REM gives `-1.0000`.
Then select ALG, retaining the chapter's real-mode and precision settings, and use the binary key between the operands:

CLEAR 7 g ÷ 4 =

This gives `3.0000`. Repeat from CLEAR with g × instead to obtain `-1.0000`.

Use MOD and REM for remainders. The keyboard's blue 1 computes a percentage, a different operation described next.

7.8 Percentages and their retained base

The three percentage operations are available on blue 1, 2, and 3. They each need two operands. Rates and percentage results are expressed in percent units: enter 12.5 for 12.5 percent, not 0.125.

Table 7.7: Percentage operand order in RPN, or first and second arguments in ALG.

Key	Operands in order	Result
g 1: %	Base Y, rate X	$YX/100$
g 2: Δ%	Original Y, new amount X	$100(X-Y)/Y$
g 3: T%	Total Y, part X	$100X/Y$

Percentage change and percentage of a total require a nonzero first operand. Negative bases and amounts follow the displayed formulas; interpret their signs in the context of the problem. Percentage change is relative to the original value, so reversing the operands generally changes both the sign and the magnitude of the answer.

In RPN, all three replace X and retain Y and every deeper register. They save both original operands in the Last registers. The retained base makes adding or subtracting a percentage amount convenient, but leaves an extra stack operand when you only wanted the percentage result. In ALG, each is simply a binary expression operation; there is no retained RPN base.

Increase an amount by 12.5 percent

Use the chapter's RPN starting state. Press:

f CLEAR 240 ENTER 12.5 g 1

The percentage amount is 30.0000 in S0 (X), with 240 retained in S1 (Y). Press + to obtain 270.0000.
For the same calculation in ALG, keep the real, precision, and display settings, with no pending prefix. Enter the base explicitly:

CLEAR 240 + (240 g 1 12.5) =

The result is again 270.0000. The group calculates the percentage amount; the first 240 supplies the amount to increase.

For percentage-change and share examples, including their RPN stack effects, see section 3.7. ALG's FUNC form uses the same operand order as table 7.7.

7.9 Factorials, permutations, and combinations

For a nonnegative integer n, factorial is $n! = 1 \times 2 \times \cdots \times n$, with $0! = 1$. Use the main $x!$ key. In real mode, the input must be an integer from 0 through 10 000. A noninteger input is rejected; use Gamma when a continuous extension is the intended calculation.

Permutations count ordered selections without replacement; combinations count selections when order does not matter:

$$nPr = \frac{n!}{(n-r)!}, \qquad nCr = \frac{n!}{r!(n-r)!}.$$

The blue Gamma function is nPr, and the blue factorial function is nCr. Enter the number of available objects n first and the number selected r second. Both must be integers satisfying $0 \le r \le n \le 10\,000$. Choosing no objects gives 1 for either operation. These are ordinary binary functions in RPN and drop the stack.

Factorials and counting results can contain many digits. The argument limit is a work bound, not a guarantee that every resulting integer fits exactly at your chosen precision. A large count may be rounded internally even though the mathematical answer is an integer. Raising working precision before calculating can preserve more of its digits.

Select three people from ten

Use the chapter's RPN starting state. For three distinct roles, order matters:

f CLEAR 10 ENTER 3 g nPr

The result is `720.0000`, since there are $10 \times 9 \times 8$ ways to fill the roles. For a group of three with no distinct roles, repeat from CLEAR using nCr:

f CLEAR 10 ENTER 3 g nCr

The result is `120.0000`. Each group was counted $3! = 6$ times in the ordered result, so $720/6 = 120$.

In ALG, use n, shifted function, r, then equals, or the explicit two-argument FUNC form described in section 4.6. Neither key assumes that a repeated selection or replacement is allowed; choose a different formula for those counting problems.

7.10 Gamma and logarithmic factorials

Gamma extends the factorial relationship beyond integer inputs: $\Gamma(n+1) = n!$ for nonnegative integers n. In particular, $\Gamma(5) = 4! = 24$, not $5!$. The main Gamma key applies $\Gamma(X)$ directly. For positive real x, one definition is

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt.$$

Gamma also has finite real values at negative nonintegers. It has poles at zero and the negative integers, and changes sign between those poles.

Orange Gamma, $\ln\Gamma$, computes the real natural logarithm of Gamma. It is finite where Gamma is finite and positive, including every positive real input and some intervals below zero. It is *not* $\ln|\Gamma(X)|$: where Gamma is negative, there is no real logarithm. For example, $\Gamma(-1/2) < 0$, so real-mode $\ln\Gamma(-1/2)$ has no finite real answer. At the poles, the Gamma and logarithmic-Gamma operations are likewise not finite evaluations. See chapter 8 for complex logarithmic values and branch conventions.

Orange factorial, $\ln x!$, calculates $\ln\Gamma(X+1)$ directly. Its real-mode input must be a finite nonnegative integer, but it does not share factorial's 10 000-input limit. Computing the logarithm directly avoids constructing the much larger factorial first. Both ln(0!) and ln(1!) are zero. Logarithmic results still have finite precision and numerical range; removing the factorial work bound does not remove those limits.

Compare factorial, Gamma, and their logarithms

Use the chapter's RPN starting state. Each row is independent and includes its clear:

Complete sequence	Display
f CLEAR 5 $x!$	120.0000
f CLEAR 5 Γ	24.0000
f CLEAR .5 Γ	1.7725
f CLEAR 5 f $\ln\Gamma$	3.1781
f CLEAR 5 f $\ln x!$	4.7875

The half-integer identity is $\Gamma(1/2) = \sqrt{\pi}$. The last two rows are $\ln 24$ and $\ln 120$, respectively.

7.11 The error function and its complement

The blue square-root function is **erf**, the error function; the blue exponential function is **erfc**, its complement:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt, \qquad \operatorname{erfc}(x) = 1 - \operatorname{erf}(x).$$

Both accept every finite real input. The error function is odd: $\operatorname{erf}(-x) = -\operatorname{erf}(x)$. At zero, erf is 0 and erfc is 1. For real inputs, erf lies between -1 and 1, and erfc between 0 and 2; sufficiently extreme inputs and finite precision can produce rounded endpoint values.

Use erfc directly when you need a small complement. Subtracting an erf value very close to 1 from 1 can lose useful digits or give zero if erf has already rounded to 1. The dedicated complementary function avoids that particular subtraction. This is a numerical reason to have both keys, even though their definitions are closely related.

Find a normal-distribution upper-tail probability

For a standard normal variable, the probability above z is $\frac{1}{2}\operatorname{erfc}(z/\sqrt{2})$. Use the chapter's RPN starting state. For $z = 2$, press:

f CLEAR 2 ENTER 2 $\sqrt{x}$ ÷ g erfc 2 ÷

The result displays as `0.0228`; its leading digits are 0.0227501319. It is a probability, approximately 2.275 percent. The first division forms $2/\sqrt{2}$; the last divides erfc by 2. The erfc key is the blue function of e^x.

Complex erf and erfc use a different calculation for nonreal inputs and have additional magnitude and convergence limits. Those restrictions do not impose a magnitude-100 bound on the real inputs described here; see chapter 8.

7.12 Constants and random numbers

Three mathematical constants have direct keys. They are generated at the current working precision, so you do not need to type a short decimal approximation.

Here e is Euler's number, the base of natural logarithms; it is not the EEX exponent-entry key. In RPN a constant commits any active number, then lifts the constant into X, preserving both Last registers. In an unfinished ALG expression, a constant inserted after an operand starts an implicit product; after a completed answer, it starts a new expression. Constants do not use FUNC.

Table 7.8: The exposed mathematical constant keys.

Constant	Sequence	Leading digits
π	f 0	3.14159265358979323846...
e	f .	2.71828182845904523536...
$\ln 2$	f ←	0.69314718055994530942...

Use pi without typing its digits

Use the chapter's ALG starting state. For the circumference $2\pi r$ of a circle of radius 3, press:

CLEAR 2 f 0 × 3 =

The constant supplies the implicit multiplication in 2π. The result is `18.8496`, in the same length unit as the radius. The calculation uses the working-precision value of pi, not 3.1416.

RAN#, the blue function of 4, generates a new real number u satisfying $0 \leq u < 1$. It uses random decimal digits at the working precision, rather than rounding a machine-size random fraction to a longer display. It takes no operand and follows the same RPN lift and ALG insertion rules as a constant. Each press requests another value; repeated values are possible, and the keyboard provides no seed-setting or repeat-sequence control.

Generate a value in the unit interval

Use the chapter's RPN starting state. Press:

f CLEAR g 4

The result is a random value at least zero and strictly less than one. There is no prescribed decimal answer to compare. Press g 4 again: the new draw occupies X and the first moves to Y. A displayed `1.0000` can be rounding of a value below 1; the interval statement concerns the stored number, not its four-place readout.

7.13 Domains, errors, and useful checks

Check the domain before treating a result as a usable real number. A negative square-root input, a nonpositive logarithm input, or an inverse-trigonometric input outside its real interval can lead to `NaN` or infinity in RPN. Other functions, such as factorial, indexed root, and arbitrary-base logarithm, issue specific input errors. ALG rejects nonfinite values when evaluating an expression. A message and a

nonfinite readout are different ways in which a calculation can fail to provide a finite answer; neither is an ordinary numerical result to carry into later work.

When a function rejects its operands, check whether an entry has already been committed before deciding how to edit it. In RPN, inspect X and Y; in ALG, inspect the expression and its trailing operand. Use the correction procedures in sections 3.8 and 4.9. LAST X and LAST Y recover saved operands after successful RPN arithmetic; they are not a general error-recovery or undo mechanism.

Near a tangent pole or a logarithm base of 1, small input errors can dominate an otherwise finite answer. Check the angle unit and operand order first, then use the precision checks in section 5.8.

Finally, selecting complex mode changes the calculation, not just its display. A complex principal root need not be the real odd root used here, and real-only operations still use real components. Before using complex mode to extend a domain, follow chapter 8 for its entry rules, principal values, and current numerical limitations.

Chapter 8

Complex Numbers

A complex number has two real components that form one value. SpicyRPN lets you enter those components directly, calculate with the pair, and view the same value in rectangular or polar form. Complex mode also extends functions such as square root and logarithm to arguments that have no finite real result.

The complex controls work in both RPN and ALG. The stack and expression rules remain those of chapters 2 and 4: a complex operand occupies one stack register or one operand position, not two. Check **COMPLEX** before calculating; displaying only the real part and calculating with both parts are different choices.

8.1 Components, controls, and the display

In rectangular form, write a complex number as

$$z = a + bi, \qquad i^2 = -1.$$

The real component is a; the imaginary component is the real coefficient b, not the whole term bi. Thus $2 - 3i$ has real component 2 and imaginary component -3. A real number has imaginary component zero. A purely imaginary number, such as $5i$, has real component zero.

With complex mode active, the main display has two rows labeled Re and Im. They show the two components of the current RPN value X, labeled S0 in the full-stack view, or the current ALG operand or result. The selected entry component has an emphasized label. Each row scrolls independently and retains its own exponent; the display settings described in chapter 5 apply to both. When a completed result changes either component, both rows return to their leading digits. Both rows belong to that one value; they do not represent separate Y and X operands. In RPN, Y is the next value at S1; each position in the full-stack view contains a complete pair. Last X and Last Y hold saved operands and are separate from the current S0/S1 values. ALG has no such stack positions.

All four complex controls are orange functions in the fixed numeric and arithmetic area. Their locations are the same in both entry modes.

Complex mode and representation are shared between RPN and ALG and survive relaunch. Changing either commits any pending numeric entry in *both* workspaces before applying the setting. It does not evaluate a whole unfinished ALG expression. If the inactive ALG workspace contains an incomplete number, such as an exponent without digits, the change is rejected. Return to ALG and finish, correct, or clear

Table 8.1: Complex controls, entered with no pending command.

Keys	Orange legend	Action
f ÷	CMPLX	Turn complex calculation and the two-row display on or off.
f 4	Re/Im	Select the other component for entry or editing.
f 5	P↔R	Switch rectangular and polar entry/display.
f 6	Re↔Im	Exchange the real and imaginary parts of the current value.

that entry, then try again. Ordinary switching between RPN and ALG preserves unfinished entries as described in chapter 4.

Press f ÷ if **COMPLEX** is absent to enable complex mode. The other three controls require it. If the rows say r and θ, press f 5 to select rectangular form. This is a representation setting, not an arithmetic conversion command taking two stack registers.

A starting state for the examples

Unless an example specifies otherwise, use 600 working digits, FIX 4, complex mode on, rectangular form, and the Scientific panel. Open PREC with g 7 to check the working and display counts. For RPN, use a fixed stack of 5 and select FIX 4 with 4 f 7; then clear with f CLEAR, the orange function of $1/x$. For ALG, select FIX 4 with f 7 4 =, then press the unshifted CLEAR. See chapter 5 for these procedures.

Start each independent sequence with no active shift or pending command. None of the examples requires empty memories. Angle units matter for polar entry and display and for trigonometric functions; the relevant examples specify them. Select degrees directly with f SIN, radians with f COS, or gradians with f TAN.

8.2 Entering and editing a complete pair

Start a new rectangular number by entering its real component. Press f 4 to select the imaginary component, then enter its coefficient. Do not enter an i, a plus sign between the components, or ENTER between them. Re/Im changes the entry focus within the same operand. It neither lifts the RPN stack nor starts a second ALG operand.

The usual digit, decimal, exponent, sign-change, and backspace keys edit the selected component. To enter a negative imaginary part, type its magnitude and press +/− while that component is being entered. During exponent entry, sign change affects

that component's exponent. Components left empty are zero, so a purely imaginary number can be entered with an explicit real zero followed by **Re/Im** and its imaginary coefficient. For scientific notation within a component, type the mantissa before **EEX**; use an explicit 1 when you mean a power of ten. A newly selected empty second component does not acquire a mantissa of 1 merely by pressing **EEX**.

In RPN, **ENTER** finishes the entire pair. As with a real entry, the first digit has already lifted the stack; selecting the second component adds no further lift. A function also finishes the pair before using it. In ALG, an operator accepts the pair as one operand, and **=** accepts it and evaluates the expression.

Multiply two complex numbers in RPN

Use the chapter's rectangular complex RPN starting state. Angle units do not affect this example. Press:

f **CLEAR** 2 **f** **4** 3 **ENTER**
4 **f** **4** 1 **+/−** ×

The first line enters $2 + 3i$; the second enters $4 - i$ and multiplies. The result is $11 + 10i$, shown as Re `11.0000` and Im `10.0000`, because $(2 + 3i)(4 - i) = 8 - 2i + 12i + 3$.

Reopening an accepted component

To change part of an existing value, use **Re/Im** before typing. With no entry active, its first press opens the current value for editing and selects the second component. A second press selects the first. In RPN this edits X in place, preserving deeper registers. Simply typing digits without first selecting a component would instead start a new operand and lift the old X.

The first digit, decimal point, or **EEX** in a selected existing component *replaces* that component's old literal. Sign change and backspace instead act on its existing literal. During a new entry, switching away from a component and back lets you continue its digits. Finish the edit with **ENTER** in RPN or **=** in ALG. An untouched rectangular component retains its full stored value and precision; it is not reconstructed from the rounded display.

Change one component without entering another operand

Use the chapter's rectangular complex RPN starting state. First enter $3 + 4i$:

f CLEAR 3 f 4 4 ENTER

Now press f 4 7 ENTER. The first digit replaces the selected imaginary component, giving Re 3.0000, Im 7.0000. Continue with:

f 4 f 4 5 ENTER

The two selections reopen the value and return focus to its real component. The result is $5 + 7i$, displayed as 5.0000 and 7.0000. The deeper registers remain zero throughout these edits.

Backspacing every character of one complex component leaves that component zero within an active pair; it does not discard the other component or undo an earlier stack lift. CLx acts on the whole operand, not just the selected component. In RPN it commits and saves the old pair in Last X, then replaces X with zero. To clear only an imaginary part, reopen it with Re/Im, enter 0, and finish the entry. CLEAR discards the whole working stack or ALG expression according to the active mode.

Complex operands in ALG

The pair-entry sequence is the same in ALG. An ordinary operator between two pairs acts on the complete operands. The expression remains visible below the two component rows and is available to VoiceOver. The component display is the current operand or result, not a running evaluation of the entire expression. While you enter a pair, the expression shows both draft components in the selected rectangular or polar form. Accepted complex operands use rectangular notation in the expression, even when the component rows show magnitude and phase.

Enter the same product in ALG

Use the chapter's rectangular complex ALG starting state. Press:

CLEAR 2 f 4 3 ×
4 f 4 1 +/− =

The result is again Re 11.0000 and Im 10.0000. To change this completed result's imaginary component to 7, press f 4 7 =. This evaluates the edited value $11 + 7i$ and updates ANS.

Re/Im after a closed group or function call reopens that operand's value for editing, including any postfix functions applied to it. The edited value replaces the entire

trailing group or call and those postfix functions, preserving the earlier expression. It does not insert multiplication or evaluate the whole expression, and ANS remains unchanged until successful = evaluation. Simply typing a new number after a closing parenthesis still starts an implicit product; component editing is an explicit replacement.

Edit a group within an unfinished expression

Use the chapter's rectangular complex ALG starting state and press:

CLEAR 10 + (2 + 3) f 4 4 =

The closed group has value 5. Re/Im selects its imaginary component, and entering 4 replaces the group with $5 + 4i$. The preceding 10+ remains, so equals gives Re `15.0000` and Im `4.0000`, and updates ANS. A closed function call is edited in the same way.

8.3 Rectangular and polar forms

Polar form describes the same number by its distance from zero and its direction in the complex plane:

$$z = r(\cos\theta + i\sin\theta), \qquad r = \sqrt{a^2 + b^2}, \qquad \theta = \operatorname{atan2}(b, a).$$

Magnitude r is nonnegative. Phase θ is measured counterclockwise from the positive real axis; a negative phase turns clockwise. The real and imaginary parts are $r\cos\theta$ and $r\sin\theta$. Figure 8.1 relates the forms for $3 + 4i$.

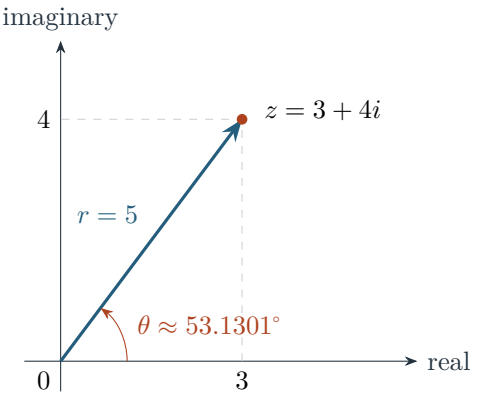


Figure 8.1: One complex value expressed by components or by magnitude and phase.

Press f 5 to change representation. The labels become r and θ, with deg, rad, or grad beside the phase. A committed value is retained as its rectangular pair; changing representation only changes how it is shown and how the next pair will be

entered. It does not move stack registers. Switching representation during entry first accepts that entry in its old representation.

Changing the angle unit while a polar pair is still being entered changes the unit in which its angle will be interpreted. It does not convert the unfinished angle's digits. Finish a polar entry before changing units when you want to preserve its direction. For committed values, changing units simply expresses the same direction in the new unit. Direct DEG/RAD/GRAD selection and the **DRG** cycle have the same effect.

To enter a polar number, select the angle unit *first*. Type the magnitude, press f 4, then type the angle. The key's legend remains **Re/Im**, but its two targets are now magnitude and angle. Accepting the pair calculates and stores its rectangular components. Reopening a component in polar form therefore edits a magnitude or phase, after which rectangular components are calculated again. Such an edit can introduce conversion rounding.

Enter a vector by length and direction

Use the chapter's complex RPN starting state. Clear the stack with f **CLEAR**. Select **DEG** with f **SIN**, and use f 5 if necessary until the display labels are r and θ. Press:

2 f 4 90 **ENTER**

The polar display shows r `2.0000` and θ `90.0000 deg`. Press f 5. The rectangular display shows Re `0.0000` and Im `2.0000`: the stored value is $2i$.

For the reverse direction, with the same RPN settings, DEG and rectangular labels, clear and enter 3 f 4 4 **ENTER**. Press f 5. The display shows magnitude `5.0000` and phase `53.1301 deg`. Press f **COS** after this completed entry to select RAD: the same value's phase displays as `0.9273 rad`.

Polar display uses the phase returned by atan2, within one half-turn on either side of zero: $-180°$ through $+180°$, or the corresponding radian or gradian limits. Ordinary entry of a negative real value gives the positive half-turn; a retained negative-zero imaginary component can select the negative endpoint. Zero is displayed with magnitude and phase both zero, although its mathematical direction is undefined.

Angles outside this displayed range are accepted and describe the same direction modulo a full turn. A negative entered magnitude is also accepted; after commitment, polar display represents the resulting value with a nonnegative magnitude and an adjusted phase. Prefer a nonnegative magnitude and the intended direction for clear entry.

8.4 Swapping components and changing signs

f 6, the Re↔Im function, changes $a + bi$ to $b + ai$. It exchanges the *rectangular* components even when polar form is showing. It does not exchange the displayed magnitude and angle. Any active pair entry is accepted first. In RPN, the value remains in X; deeper registers and both Last registers are unchanged. In ALG it replaces the current operand, or starts an edited operand from a completed result; press = to make that edited value the answer.

For example, use the chapter's rectangular complex RPN starting state. After f CLEAR 2 f 4 3 ENTER, press f 6. The two rows become `3.0000` and `2.0000`, representing $3 + 2i$. Repeating the swap restores $2 + 3i$. This is separate from RPN's $x \leftrightarrow y$, which exchanges two complete stack operands.

Sign change also depends on whether entry is active. While editing, +/− changes the selected component's sign, or its exponent's sign when an exponent is being entered. With no entry active, it negates the *whole* complex number, changing both a and b. To form the conjugate $a - bi$ of an accepted rectangular value, reopen its imaginary component and replace it with the negative of its coefficient, then accept the edit. For a simple component such as 3 with no exponent in its editable literal, f 4 +/− ENTER also changes it to −3 in RPN. In ALG, edit a completed result and finish with =. If a reopened component's literal contains an exponent, sign change affects that exponent instead; re-enter the desired signed coefficient when necessary. Component selection and swapping are editing controls; they are not general undo operations.

8.5 Arithmetic and supported scientific functions

With **COMPLEX** showing, ordinary arithmetic uses both components. Addition and subtraction combine corresponding parts; multiplication uses $i^2 = -1$. Division uses a nonzero complex denominator. In RPN the order remains $Y - X$, Y/X, and Y^X, with the usual stack drop. Unary functions replace X. In ALG the same functions follow the precedence and argument-entry rules in chapter 4.

Both entry modes use the same complex functions. Valid real-axis results retain real-function behavior, including exact trigonometric values at a quadrant in DEG or GRAD. Practical limits appear in section 8.9; section C.4 describes the numerical libraries and rounding conventions.

Constants and random-number generation supply real values with zero imaginary part. They supply a whole new value, not just the selected component of an open pair. Storage and recall copy complete complex values when CMPLX is on; with it off, they write only the real component. Arithmetic storage follows the same complex setting. See chapter 6 for their operand and stack effects, and chapter 7 for the scientific keys and formulas.

Table 8.2: Scientific operations that use the complete complex value.

Function family	Complex behavior
Arithmetic, sign, reciprocal	Add, subtract, multiply, divide, negate, and take $1/z$.
Square, square root, power, indexed root	Square uses z^2. Other roots and powers select one principal result; see section 8.8.
ABS	Return the real magnitude $\lvert z\rvert$, with zero imaginary part. It is the orange function of $\times$.
Exponentials and logarithms	e^z, 10^z, natural, common, binary, and base logarithms. Logarithms use the branch conventions below.
Trigonometric and inverse trigonometric	SIN, COS, TAN and their blue inverses use the selected angle unit for both components of the argument or result.
Hyperbolic and inverse hyperbolic	HYP followed by SIN, COS, or TAN, and orange HYP followed by those keys. These use dimensionless complex arguments, independent of DEG/RAD/GRAD.
Gamma and factorial families	$\Gamma(z)$, logarithmic Gamma, $z! = \Gamma(z+1)$, and logarithmic factorial.
erf and erfc	Complex error function and its complement, subject to the nonreal-argument limits in section 8.9.
%, $\Delta\%$, **T%**	Their algebraic formulas extend to complex operands. RPN still retains Y and all deeper positions.

Angles in a complex function

For a complex trig argument $a + bi$ in DEG, the calculator evaluates the radian function at $(a + bi)\pi/180$. Both components are scaled; the imaginary component is not left in radians. In GRAD the scale is $\pi/200$. An inverse trig function scales both components of its radian answer back to the selected unit. Choose RAD for the usual mathematical complex-function formulas without an additional scale.

Logarithms and exponentials do not use the angle setting for their rectangular components. In particular, the imaginary part of $\log z$ is an angle in radians, even with DEG selected. Likewise, e^{a+bi} uses b in radians. If the result is then shown in polar form, the *phase of that result* is displayed in the selected angle unit. Do not confuse it with a rectangular imaginary component that happens to encode an angle.

A complex sine with degree-scaled components

Use the chapter's rectangular complex RPN starting state. Select DEG with f SIN, then press:

f CLEAR 90 f 4 30 SIN

This calculates $\sin(\pi/2 + i\pi/6) = \cosh(\pi/6)$. The real component displays as `1.1402`; the imaginary component is zero. Reducing the real angle by exact quarter turns preserves this axis value before the remaining complex angle is converted to radians. For real-axis trigonometric inputs, the calculator uses the real-function path when it applies.

8.6 Turning complex mode off

Pressing f ÷ with **COMPLEX** present returns to normal real-number calculation. The two-row display closes and only the real component is shown. The imaginary components already stored in stack values, memories, and saved operands remain intact until those values are replaced. Turning complex mode back on reveals the retained components. The rectangular/polar preference is also retained.

Every new value written with complex mode off is real: calculations, stores, recalls, new Last-register captures, and an idle ENTER copy all write an imaginary component of zero. In particular, real-mode RCL does *not* place a hidden imaginary component in X. It leaves the source memory unchanged, so a later recall with CMPLX on can still retrieve that memory's complete pair. A real-mode STO replaces the destination with a real value; arithmetic STO computes the new destination from real components.

Lifting, dropping, swapping, or rolling existing stack values moves their retained components together. An idle ENTER moves the original X to Y but creates a real copy in X when CMPLX is off. Turning CMPLX on or reopening the application does not reconstruct a component that a new assignment has replaced with zero.

Keep a complex value while calculating in real mode

Use the chapter's rectangular complex RPN starting state. This example deliberately replaces ordinary register R0. Enter:

f CLEAR 2 f 4 3 STO 0

Both X and R0 contain $2 + 3i$. Press f ÷ to turn complex mode off, then RCL 0. Recall puts $2 + 0i$ in X, displayed as `2.0000`; it leaves R0 unchanged. Press $1/x$. The result is `0.5000`, calculated from the real 2.
Turn complex mode on again with f ÷. X now shows $0.5 + 0i$. Press RCL 0: the recalled pair is still $2 + 3i$, displayed as `2.0000` and `3.0000`. The real-mode recall and reciprocal changed working values, not R0. Had you overwritten R0 with CMPLX off, its imaginary component would instead have been replaced by zero.

The real-only interpretation also affects unfinished ALG expressions when they are subsequently evaluated. The complex setting is shared; an expression can contain retained complex operands while real mode is selected. Restore complex mode before evaluating if the intended calculation needs both components.

Associative memory always uses the real component as its key, in both modes. Keys $3 + 4i$, $3 - 8i$, and 3 select the same entry; storing again replaces that entry. Its value follows the same store/recall rule as ordinary memory: both components with CMPLX on, only the real component with CMPLX off. See section 6.7.

8.7 Functions that use only real components

Complex mode does not extend every operation to a complex domain. The operations below take the real components of their inputs and return real results. A nonzero imaginary input normally produces the status warning `Used real portion only`. The warning is information about the calculation, not an instruction to reinterpret the displayed answer as complex.

These operations are available in both entry modes. ALG statistics and multi-input financial calculations use labeled worksheets; RND acts directly on a completed expression or single entered number. A magnitude is different from a real-part projection: ABS of $3 + 4i$ returns 5, while a real-only function starts from 3. Use polar display to inspect a single pair's phase; the two-operand atan2 function has the different input arrangement described in chapter 7.

Table 8.3: Operations with real-only inputs even when CMPLX is active.

Operations	Interpretation
INT, FRAC, NINT, floor, ceiling	Apply the stated integer or fractional rule to $\operatorname{Re} z$; they do not operate separately on both components.
MOD and REM	Compute the respective remainders of the real components of both operands.
nPr and nCr	Require real components satisfying $0 \le r \le n \le 10\,000$, with both integers.
atan2	Treat the two operands' real parts as the vertical and horizontal coordinates. It does not take the phase of one pair.
Unit conversions	Convert the real component and produce a real result, including when source and destination units are equal.
Statistics, finance, dates, and RND	Use real quantities. Stored pairs can remain in their banks until a command replaces them; their imaginary parts do not become statistical or financial inputs.

For example, use the chapter's rectangular complex RPN starting state. Press f CLEAR 1.2 +/− f 4 9 g 6. The blue function of 6 is floor. It returns Re -2.0000, Im 0.0000, and reports that only the real portion was used. It has not floored the imaginary 9 as a second result.

Read the command's domain before using a stored complex value in a real-only task. The absence of a current warning does not certify a bank's contents: later entry can clear a previous warning, and ordinary storage/recall is deliberately allowed to retain complete pairs.

8.8 Principal results and branch conventions

Many complex equations have more than one solution. A nonzero number has two square roots, and logarithms differ by integer multiples of $2\pi i$. The calculator returns one selected result. It does not list all roots or carry a branch-selection setting.

For the principal logarithm, away from its negative-real cut,

$$\operatorname{Log} z = \ln|z| + i \operatorname{Arg} z.$$

On the negative real axis, logarithm explicitly chooses the upper side: $\operatorname{Log}(-r) = \ln r + i\pi$ for $r > 0$, including an argument whose imaginary component is negative

zero. Square root likewise chooses $\sqrt{-r} = +i\sqrt{r}$. Away from that cut it returns the square root with nonnegative real part and an imaginary sign following the input's imaginary sign.

General powers follow $z^w = \exp(w \operatorname{Log} z)$, evaluated by MPC's power function. An indexed root uses $z^{1/w}$, with Y as the radicand and X as the index in RPN. The index must be nonzero and may itself be complex. Consequently, a negative real radicand with an odd integer index gives the principal complex root in complex mode. Real mode instead supplies the real odd root.

A root and a logarithm on the negative real axis

Use the chapter's rectangular complex RPN starting state. Each row is an independent calculation; first press f CLEAR. The angle setting does not change these rectangular results. LN is the orange function of e^x; indexed root is the orange function of y^x.

Keys after clearing	Re	Im
4 +/− $\sqrt{x}$	0.0000	2.0000
1 +/− f LN	0.0000	3.1416
8 +/− ENTER 3 f $\sqrt[x]{y}$	1.0000	1.7321

The last result is $1+i\sqrt{3}$, one cube root of -8. It is different from the real-mode result -2.

Common and binary logarithms equal $\operatorname{Log} z / \ln 10$ and $\operatorname{Log} z / \ln 2$, respectively; MPC evaluates each directly. The two-operand base logarithm calculates

$$\operatorname{Log} X / \operatorname{Log} Y.$$

Zero arguments and a base with zero logarithm do not give finite logarithmic results. Familiar real identities involving roots, powers, or logarithms cannot always be rearranged freely across a complex branch cut. For example, $\sqrt{z^2}$ need not equal z.

Inverse functions and logarithmic Gamma

Inverse trig and inverse hyperbolic functions also select branches. For real arguments where a finite real inverse is available, that real result is retained. Exact values on a branch cut require particular care: matching a function name on another calculator does not guarantee matching imaginary signs there. The table gives representative boundary values; section C.4 supplies the full identities and their numerical interpretation.

To check one of these signs, use the chapter's rectangular complex RPN starting state. Select RAD with f COS, then press f CLEAR 2 g SIN. Inverse

Table 8.4: Selected cut values in radians. Let $A = \ln(2+\sqrt{3})$ and $B = \frac{1}{2}\ln 3$. These are exact-axis conventions of this version.

Function and argument	Selected value
$\arcsin(2)$, $\arcsin(-2)$	$\pi/2 - iA$, $-\pi/2 + iA$
$\arccos(2)$, $\arccos(-2)$	iA, $\pi - iA$
$\operatorname{atanh}(2)$, $\operatorname{atanh}(-2)$	$B - i\pi/2$, $-B + i\pi/2$
$\operatorname{acosh}(-2)$	$A + i\pi$
$\arctan(2i)$	$\pi/2 + iB$
$\operatorname{asinh}(-2i)$	$-A - i\pi/2$

sine returns Re `1.5708` and Im `-1.3170`. Under DEG, both components of this inverse-trigonometric result would be scaled to degrees; inverse hyperbolic functions remain independent of the angle setting.

For a nonreal argument, logarithmic Gamma is calculated as the principal logarithm of the computed $\Gamma(z)$. Logarithmic factorial similarly takes the logarithm of $\Gamma(z+1)$. These are not an unwrapped analytic log-Gamma whose imaginary part can grow beyond the principal-log interval. On real inputs the available real-function path is used where valid; otherwise the complex extension is attempted. Complex mode therefore permits factorials of nonintegers, such as $\Gamma(3/2)$ for 0.5!, but does not remove Gamma's poles at nonpositive integers or the factorial poles at negative integers. The real-mode factorial limit of 10 000 is not a general restriction on this Gamma extension.

8.9 Numerical limits and interpreting a result

Complex Gamma, factorial, and error functions use approximation or series algorithms. In particular, nonreal ERFC is formed by subtracting erf from 1, so a small tail can lose digits; real ERFC uses the direct calculation described in section 7.11. The numerical methods and their rounding limits are documented in section C.4.

For nonreal inputs, erf and erfc have an approximate magnitude limit of 100 and can report failure to converge. Real arguments use the real error-function path instead. Nonreal Gamma calculations support the exposed working precision range through 10 000 digits and can become expensive as precision increases. These are calculation limits, not limits on the number of places the display can scroll.

Complex mode also leaves genuine singularities: division by zero, logarithm at zero, poles of Gamma, and inverse-function singularities are not made finite by adding a second component row. RPN can display `NaN` or infinity for a nonfinite arithmetic result; ALG rejects nonfinite values during evaluation. Other operations report a domain or convergence message. A nonfinite current value cannot be reopened with **Re/Im**; clear or replace it before editing components.

When a result is unexpected, check **COMPLEX**, rectangular versus polar labels, the selected component, and the angle unit first. Then check whether the function is real-only, whether a principal branch explains the sign, or whether a special-function limit applies. For precision checks, use section 5.8.

Chapter 9

Unit Conversions

The conversion tool changes the numerical value of a quantity from one unit to another. Enter a length in inches, for example, choose inches as the source and metres as the destination, and apply the conversion. The result is ready for the next calculation in either RPN or ALG.

SpicyRPN provides fifteen quantity families, from length and temperature to frequency and torque. Units are selected within a family; the tool does not infer a unit from a number or carry units through later arithmetic. You remain responsible for interpreting the result in the destination unit and combining compatible quantities.

9.1 Choose the quantity and conversion direction

Conversions use the real component

The conversion tool uses the real component of its input and returns a real result. It does not scale both components of a complex number, and a polar display does not make the displayed magnitude the input. With complex mode active, a nonzero imaginary component produces an imaginary-data warning. With complex mode off, a hidden imaginary component is also omitted from the replacement value. Preserve the original first if both components matter; see chapter 8.

Enter the number on the calculator before opening the conversion panel. There is no separate value field in Convert X. In RPN the input is X, labeled S0 in the full-stack view, including a number still being typed. Y is the next value at S1. In ALG the input is the current operand or completed result, as explained in section 9.2. Opening the panel does not finish a number or evaluate an expression.

With no pending command or active shift, press g 8, the blue CONV function. The panel is titled Convert X in both entry modes. It has three menu controls:

Select Dimension first, then check both unit menus. Changing the dimension replaces both selections with that family's first and second listed units. Changing to Length, for example, selects meter to kilometer. A new calculator state initially offers Length, meter to foot; later visits retain your selected choices. Always read From and To before applying, especially after changing the dimension.

Table 9.1: The conversion panel's controls.

Control	What to select
Dimension	The kind of quantity, such as Length, Volume, or Temperature.
From	The unit in which the current number is expressed.
To	The unit in which you want the result.
Apply	Perform the conversion and close the panel if it succeeds.
Close	Return to the calculator without converting the number.

Each menu entry includes a name and symbol, such as inch (in) or US gallon (US gal). The unit menus contain only units belonging to the chosen dimension. To reverse a conversion, select the old destination under From and the old source under To; there is no separate swap control.

Close leaves the current number, unfinished entry, or expression untouched. Changes to the menu choices are retained even if you close without applying. For example, closing midway through an RPN entry `12.3400` leaves those characters available for further typing; it does not turn the entry into a completed result.

If applying reports an error, the panel stays open and shows a message. Close it to correct the calculator input, then reopen it and apply the intended units. An unfinished ALG exponent, for example, must be completed before conversion can succeed. A nonfinite conversion result is rejected.

A starting state for the examples

Unless an example says otherwise, use real mode, 600 working digits, and FIX 4; RPN examples use a fixed stack of 5. Follow the setup procedures in sections 2.2, 5.2 and 5.4. Start with no open tool, active shift, or pending command. Each independent sequence includes its clear, and no example requires empty memories.

9.2 What changes in RPN and ALG

RPN: replace X in place

A successful conversion finishes an active number entry and replaces X with the converted value. It does not lift or drop the stack. Y and all deeper positions stay where they are, in both fixed and unlimited stacks. If you were typing the input, the lift that began that entry has already occurred; applying the conversion does not add another one.

Both Last X and Last Y, the saved operands separate from the current S0/S1 values, are preserved. This differs from an ordinary single-operand mathematical function,

which saves operands as described in section 2.9. Consequently, **LAST X** does not recover the input of a conversion merely because it was the most recent operation. Store or duplicate an input before conversion if you need to retain it.

Convert one operand while another waits

Use the chapter's RPN starting state. Clear with **f** **CLEAR**, the orange function of $1/x$, then enter:

2 **ENTER** 12 **g** **8**

In Convert X, choose Dimension: Length, From: inch (in), and To: foot (ft), then press **Apply**. The main S0 (X) display is `1.0000`; S1 (Y) still holds `2.0000`. You can check it in the full-stack view. Close that view if open. Interpreting the waiting 2 as feet, press **+**. The sum is `3.0000` feet: $2\,\text{ft} + 12\,\text{in} = 3\,\text{ft}$.

After applying, a new digit starts a new RPN number and lifts the converted result in the usual way. An idle **ENTER** duplicates the converted result. You do not need an extra **ENTER** merely to finish the conversion.

ALG: replace the current operand

In an unfinished ALG expression, conversion replaces the current trailing operand and preserves the surrounding expression. It does not evaluate the whole expression. A closed parenthesized group or function call counts as one operand: the converted value replaces that group or call. Supply the operand you intend to convert before opening the panel; an expression ending in an operator is not a request to convert its eventual total.

Convert centimetres inside an ALG expression

Use the chapter's ALG starting state. Press the unshifted **CLEAR**, then enter:

2 **+** 100 **g** **8**

Choose Dimension: Length, From: centimeter (cm), and To: meter (m), then press **Apply**. The current operand becomes `1.0000`; the expression now contains $2 + 1$. Continue with:

× 3 **=**

The result is `5.0000`. Interpreting the initial 2 as metres and 3 as a dimensionless multiplier, the calculation is $2\,\text{m} + (100\,\text{cm}) \times 3 = 5\,\text{m}$. The conversion affected the 100, not the sum of 2 and 100.

To convert an entire ALG calculation, finish it with = first. Applying the conversion then starts a new expression containing only the converted value. The previous ANS remains unchanged until you press = again. You may instead continue with an operator and further operands, then evaluate that new expression. See section 4.8 for answer and continuation rules.

Conversions do not alter memory banks or the other entry mode's suspended workspace. A recalled value is converted in the working area; its source memory remains unchanged unless you store the new value there yourself. No unit tag is attached to the resulting number, so remember its meaning when storing or reusing it.

9.3 Length, area, volume, and mass

Choose the dimension of the value you actually have. A floor area measured in square feet belongs to Area, not Length. A container's capacity in cubic feet belongs to Volume. The area and volume conversions already include the squared or cubed length factors: you should not square or cube the numerical input again.

For example, one foot is 0.3048 metre, one square foot is 0.09290304 square metre, and one cubic foot is 28.316846592 litres. The three factors are different because the quantities are different. A litre is 1000 cm^3, while a millilitre and a cubic centimetre represent the same volume.

Table 9.2: Representative size and mass choices. The full selectable inventory is in section 15.10.

Dimension	Examples and distinctions
Length	meter (m), inch (in), foot (ft), mile (mi), nautical mile (nmi). Use a length unit for a distance, not an area or capacity.
Area	square meter (m^2), square foot (ft^2), acre (acre), hectare (ha). The factor already includes the square.
Volume	liter (L), cubic centimeter (cm^3), US gallon (US gal), Imperial gallon (Imp gal). Read liquid-unit prefixes carefully.
Mass	kilogram (kg), ounce (oz), pound (lb), metric tonne (t), US short ton (short ton), Imperial long ton (long ton). Mass ounces are distinct from fluid ounces.

The inch, foot, yard, and mile use the international length definitions: one inch is 0.0254 metre, one foot is 0.3048 metre, and one mile is 1609.344 metres. The nautical mile is 1852 metres. No separate survey-foot or historical-mile choice is provided. For land area, an acre is 4046.8564224 square metres and a hectare is 10 000 square metres.

Mass uses the ordinary avoirdupois ounce and pound: 16 ounces make one pound, and one pound is 0.45359237 kilogram. A stone is 14 pounds. The ounce in Mass is not a fluid ounce, and there is no troy-ounce choice. A mass cannot be converted directly to a volume without additional information such as density.

Check an area, a capacity, and a mass

For each independent row below, use the chapter's RPN starting state. Press f CLEAR, type the input, then press g 8. Choose the stated dimension and the named From and To units, then press Apply.

Input	Dimension; From ⟶ To	Display
1	Area; acre ⟶ square meter	4,046.8564
2.5	Volume; liter ⟶ cubic centimeter	2,500.0000
180	Mass; pound ⟶ kilogram	81.6466

The results are in square metres, cubic centimetres, and kilograms, respectively. The unrounded mass result is 81.6466266 kilograms. The display rounds it; the retained value is used by later arithmetic.

9.4 US and Imperial names are part of the unit

US and Imperial liquid measures share familiar names but have different sizes. Select the full name, including its prefix. The US entries in Volume are the liquid measures; they are not US dry pints, quarts, or gallons. Table 9.3 gives the definitions used by the tool.

Table 9.3: Litres in one US or Imperial liquid unit. These are the defining decimal factors, before working-precision rounding.

Unit	US	Imperial
Fluid ounce	0.0295735295625	0.0284130625
Pint	0.473176473	0.56826125
Quart	0.946352946	1.1365225
Gallon	3.785411784	4.54609

Both systems have two pints to a quart and four quarts to a gallon, but a US pint contains 16 US fluid ounces while an Imperial pint contains 20 Imperial fluid ounces. An Imperial gallon is larger than a US gallon, although an Imperial fluid ounce is smaller than a US fluid ounce. Applying one blanket US-to-Imperial multiplier to every similarly named unit would therefore be wrong.

The mass menu also distinguishes three large units. A metric tonne is 1000 kilograms. A US short ton is 2000 pounds, or 907.18474 kilograms; an Imperial long ton is 2240 pounds, or 1016.0469088 kilograms. A quantity labeled only "tons" needs its convention identified before conversion. None of these choices changes the definition of the pound.

Compare the two gallons

Use the chapter's RPN starting state. Press f CLEAR 1 g 8. Select Dimension: Volume, From: US gallon (US gal), and To: liter (L), then press Apply. The display is `3.7854` litres.
Repeat the complete clear-and-entry sequence with a fresh 1. This time select From: Imperial gallon (Imp gal) and keep To: liter (L). After Apply, the display is `4.5461` litres. Re-entering 1 matters: simply reopening the tool after the first result would convert the current 3.785411784, not the original one gallon.

9.5 Temperature readings and temperature differences

Temperature offers kelvin (K), degree Celsius (°C), degree Fahrenheit (°F), and degree Rankine (°R). These choices convert temperature *readings*, including their different zero points. If C, F, K, and R are the numerical readings in those units, the defining relationships are

$$K = C + 273.15, \qquad K = (F + 459.67)\frac{5}{9},$$
$$K = R\frac{5}{9}, \qquad F = \frac{9}{5}C + 32.$$

The tool applies the appropriate scale and offset automatically. You do not enter 273.15 or 32 as a second operand.

Convert a Fahrenheit reading

Use the chapter's RPN starting state. Press:

f CLEAR 68 g 8

Choose Dimension: Temperature, From: degree Fahrenheit, and To: degree Celsius, then press Apply. The display is `20.0000`. The check is $(68 - 32) \times 5/9 = 20$: 68 degrees Fahrenheit is 20 degrees Celsius.

A temperature *difference* has no zero-point offset. A rise of 10 Celsius degrees is a rise of 18 Fahrenheit degrees; converting the reading 10 degrees Celsius with the tool instead gives a reading of 50 degrees Fahrenheit. There is no separate temperature-difference menu. For an interval, multiply by 9/5 from Celsius or

kelvin increments to Fahrenheit or Rankine increments, or by 5/9 in the reverse direction. Celsius and kelvin increments have equal size, as do Fahrenheit and Rankine increments.

For instance, in the preceding RPN settings with no pending prefix, f CLEAR 10 ENTER 9 × 5 ÷ displays 18.0000, the Fahrenheit interval corresponding to a 10-degree Celsius change. This arithmetic deliberately omits the offset.

The tool applies its formulas to negative inputs too. It does not reject a reading below absolute zero or certify that a temperature is physically attainable. Around a mathematical zero, finite-precision offset arithmetic can leave a very small nonzero residue; see section 9.9.

9.6 Motion, force, pressure, and torque

The mechanical families use scale factors without temperature-style offsets. Choose the complete compound unit from its own dimension: speed already includes time in the denominator, acceleration includes time squared, and pressure includes area. There is no need to convert each constituent unit separately.

Table 9.4: Representative mechanical choices and the quantities they express.

Dimension	Examples and distinctions
Speed	meter per second (m/s), mile per hour (mph), knot (kn). These express distance per unit time.
Acceleration	meter per second squared (m/s^2), standard gravity (g_0). Do not confuse g_0 with gram (g).
Force	newton (N), pound-force (lbf). A pound-force is not a mass.
Pressure	pascal (Pa), kilopascal (kPa), pound per square inch (psi), torr (Torr). Conversion preserves the input's pressure reference.
Torque	newton-meter ($N \cdot m$), pound-foot ($lbf \cdot ft$). Choose Torque even though energy can have the same dimensions.

A knot is one nautical mile per hour, or 1.852 kilometres per hour. One mile per hour is 0.44704 metre per second. Standard gravity is the fixed reference $g_0 = 9.80665\,m/s^2$; it does not estimate the local acceleration of gravity. Its symbol g_0 also differs from g, the gram in the Mass menu.

Keep pound mass and pound-force distinct. The Mass menu's pound converts to kilograms; the Force menu's pound-force converts to newtons, using $1\,lbf = 4.4482216152605\,N$. The tool does not turn a mass into a force by selecting a local gravity. To calculate weight from mass, first express the mass and acceleration in compatible units, then multiply them.

Pressure conversion changes the unit, not its reference zero. A gauge-pressure input remains a gauge-pressure number in the destination unit; the tool does not add atmospheric pressure to make it absolute. One standard atmosphere is 101 325 pascals, one bar is 100 000 pascals, and 760 torr is one standard atmosphere under the catalog's definition.

Torque's pound-foot means pound-force times foot. The factor is $1\,\text{lbf} \cdot \text{ft} = 1.3558179483314004\,\text{N} \cdot \text{m}$. Torque and energy remain separate dimensions in the menus even though both can involve a force times a distance. A torque conversion is not a calculation of work through an angle.

Convert a pressure to kilopascals

Use the chapter's RPN starting state. Press f CLEAR 35 g 8. Select Dimension: Pressure, From: pound per square inch (psi), and To: kilopascal (kPa), then press Apply. The display is `241.3165` kilopascals. If the original 35 psi was gauge pressure, this result is also gauge pressure.

9.7 Energy, power, time, and frequency

Energy and power answer different questions. Power is a rate of energy transfer; multiplying a constant power by its duration gives an energy. The conversion tool can convert watts to horsepower or kilowatt-hours to joules, but cannot convert watts directly to joules without a duration.

Table 9.5: Representative energy, power, time, and frequency choices.

Dimension	Examples and distinctions
Energy	joule (J), kilowatt-hour (kWh), thermochemical calorie (cal), International Table BTU (BTU). Identify the named calorie or BTU convention.
Power	watt (W), kilowatt (kW), mechanical horsepower (hp). A duration is needed to obtain energy.
Time	second (s), hour (h), day (d). Use decimal quantities; a day is a fixed duration of 24 hours.
Frequency	hertz (Hz), kilohertz (kHz), gigahertz (GHz). Frequency and period are reciprocals, not units in one family.

One watt-hour is 3600 joules and one kilowatt-hour is 3.6 megajoules. The calorie choices use the thermochemical definition: $1\,\text{cal} = 4.184\,\text{J}$ and $1\,\text{kcal} = 4184\,\text{J}$. A nutrition label's "Calorie" convention ordinarily means a kilocalorie, so identify the source convention before choosing cal or kcal. The BTU choice is specifically the International Table unit, using 1055.05585262 joules per BTU. The power choice is

mechanical horsepower, using 745.69987158227022 watts per hp; other horsepower conventions are not separate choices.

Convert energy calculated from power and time

Use the chapter's RPN starting state. A constant 1.5-kilowatt load runs for 3 hours. Form the kilowatt-hours first:

f CLEAR 1.5 ENTER 3 ×

The display is `4.5000`, representing 4.5 kWh. Press g 8; choose Dimension: Energy, From: kilowatt-hour (kWh), and To: megajoule (MJ), then press **Apply**. The display is `16.2000`, since $4.5 \times 3.6 = 16.2$ megajoules.

Time uses fixed durations: 60 seconds per minute, 3600 seconds per hour, and 86 400 seconds per day. A day here is exactly 24 hours; the tool does not account for calendar dates, time zones, or changes in civil-clock time. Use the date operations in chapter 13 for calendar-date calculations.

Time inputs are decimal quantities, not hours-and-minutes fields. For example, 1.5 hours means one hour and thirty minutes; 1.30 hours means one hour and eighteen minutes. Convert mixed units to a common unit before adding them.

Frequency units differ by powers of 1000: 1 GHz = 1000 MHz, for example. A period in seconds and a frequency in hertz are reciprocal quantities, not two units within one conversion family. Use $1/x$ to take a reciprocal when that relationship is appropriate, then convert the resulting frequency or duration within its own family.

9.8 Angle conversion and the angle setting

The Angle dimension offers radian (rad), degree (°), gradian (grad), and turn (turn). Their defining full-circle relation is

$$1\,\text{turn} = 360^\circ = 400\,\text{grad} = 2\pi\,\text{rad}.$$

Enter decimal degrees, not a packed degrees-minutes-seconds value. An angle of 30°30′, for example, is 30.5 decimal degrees. The tool scales the number without reducing it to one revolution: multiple turns and negative angles retain their meaning.

The From and To selections determine the conversion regardless of the display's **DEG**, **RAD**, or **GRAD** setting. Applying a conversion does not change that setting. Conversely, changing the angle setting does not convert a scalar number already in the working area. For a committed complex pair in polar display, it re-expresses the same phase in the new unit. During unfinished polar entry, however, it changes

how the unchanged angle digits will be interpreted. Finish the pair before changing units to preserve its direction; see section 8.3.

Convert degrees, then use the radian result

Use the chapter's RPN starting state with **DEG**. On the Scientific panel, select degrees if needed with f SIN, its orange DEG function. Press:

f CLEAR 90 g 8

Choose Dimension: Angle, From: degree, and To: radian, then press Apply. The display is `1.5708`, representing $\pi/2$ radians. The flag still says **DEG**. Press f COS, the orange RAD function, then SIN. The display is `1.0000`. Selecting RAD made sine interpret the converted operand as radians; it did not change that operand. The undisplayed digits of $\pi/2$ were retained for the sine calculation.

In ALG, choose the angle setting before entering a trigonometric function. A function already inserted in an expression retains the angle unit selected at its insertion, as described in section 4.5. Converting an operand does not rewrite that function's angle convention.

9.9 Conversion factors, precision, and rounding

Except for temperature offsets and the angle path through radians, conversion is a change of scale. If s_{from} and s_{to} express one source and destination unit in a common base unit, the numerical result is

$$v_{\text{to}} = v_{\text{from}} \frac{s_{\text{from}}}{s_{\text{to}}}.$$

Temperature first accounts for the source zero point and finally removes the destination offset. The separate treatment explains why zero normally stays zero for a scale conversion, but zero Celsius becomes 273.15 kelvin.

Conversion factors are stored as decimal definitions or ratios of integers and evaluated at the current working precision. For example, the inch factor is 254/10000 metre; the torr factor is 101325/760 pascal. Angles use π computed at the active precision. The tool does not pass values through ordinary machine-precision decimal factors before doing the calculation.

An exact unit definition does not make every floating-point result exact. Decimal factors such as 0.0254, repeating quotients, and irrational multiples of π generally need finite binary approximations. Each real arithmetic step rounds to nearest, with ties to even, at the selected working precision. A sequence of such steps can accumulate rounding; the whole conversion is not a promise of one perfectly rounded exact formula.

What the displayed digits mean

Conversions retain undisplayed digits under the formatting rules in section 5.6. Even choosing the same unit in both menus creates a result at the current working precision; it does not recover information lost from an older input. See section 5.7 when changing precision.

A near-zero temperature result

Converting the reading 32 degrees Fahrenheit to Celsius mathematically gives zero. The finite-precision sequence through kelvin can leave a tiny positive or negative residual after subtracting the offset. FIX may show that residual in scientific notation because it avoids displaying a nonzero value as zero. The residue is arithmetic rounding, not a different freezing-point definition. Read the exponent and judge the absolute size of the discrepancy; repeating from the original input at higher working precision can reduce it.

A conversion out and back likewise need not restore the original binary value bit for bit. Prefer converting directly from the original quantity to the required unit, and retain the original when comparing several destinations. A useful check is to convert a familiar reference quantity, such as 12 inches to 1 foot or 1 kilowatt-hour to 3.6 megajoules, using the same selected units and direction.

The tables in this chapter highlight choices that are easy to confuse. For every selectable unit, use section 15.10; for the complete factor and offset catalog, use section C.13.

Part III

Statistical and Financial Calculations

Chapter 10

Statistics

SpicyRPN accumulates observations and calculates averages, sample standard deviations, and a straight-line fit. Enter each observation once, then request as many results as you need from the stored totals. You can add another observation or remove a known incorrect one without starting the whole calculation again.

In RPN, three statistics keys occupy the fixed row beside STO and RCL. In ALG, orange FUNC, labeled STAT, opens labeled worksheets. Both modes use the same six accumulators, retained across mode changes and relaunch. Neither mode stores the original sample list, so removing an observation requires its original coordinates.

RPN readers can follow the key sequences below. ALG readers should start with section 10.10 for worksheet entry, then use the sections on means, deviations, regression, and estimation for the shared formulas and interpretation. Worksheet fields replace the RPN entry sequences.

In the RPN instructions, X is the current stack position labeled S0 in the full-stack view, and Y is S1. Last X and Last Y are separate saved operands. Lowercase x, y name an observation's coordinates; ALG's X and Y fields are named worksheet inputs, not RPN stack positions.

10.1 Prepare the statistical bank

The statistical bank is separate from the stack, ordinary memories, and financial registers. Clearing the stack does not remove observations. Before a new data set, clear the statistics explicitly with f $\Sigma+$, selecting the orange CL_{Σ} legend. This removes all six accumulators and turns off **STAT**. It leaves the committed stack, both Last registers, ordinary memories, and financial data intact.

The three keys and their alternate functions are collected in table 10.1. Press the indicated shift and then the main key; an alternate legend is not a separate button.

For this chapter's examples, use RPN, real mode (**COMPLEX** off), a fixed stack of 5, 600 working digits, and FIX 4. The setup procedures are in sections 3.1 and 5.2. Angle units, payment timing, date order, and previous financial-register contents do not affect these examples. Begin with no active shift, pending command, or open tool. Each independent data set below starts with f $\Sigma+$ to clear statistics and f **CLEAR** to clear the stack; CLEAR is the orange function of $1/x$.

Table 10.1: The RPN statistics row.

Main key	Orange function	Blue function
$\Sigma+$: add sample	CL_Σ: clear statistics	$\Sigma-$: remove sample
$\bar{x}$: means	Weighted mean, labeled $\overline{xw}$	**SD**: sample standard deviations
L.R.: linear regression	$\hat{y}$: estimate and correlation	RCL_Σ: copy accumulators

Clearing with an unfinished entry

The dedicated CL_Σ command first finishes a number that is still being entered, then clears the bank. It does not erase that number. To preserve the unfinished entry exactly as typed, use the statistics group of **MCLR**; its selectors and cancellation rules are in section 6.10. Both methods leave earlier accumulator copies in ordinary memory intact.

10.2 Enter observations in the right order

Adding or removing a pair requires both X (S0) and Y (S1) positions; a single-position unlimited stack needs another operand first. Both $\Sigma+$ and $\Sigma-$ require finite real components in X and Y. NaN or either infinity produces Statistics samples must be finite. The commands also check all six existing totals and all six proposed totals; a nonfinite total, including overflow while updating the sums, produces Statistics totals must be finite. Either error leaves the statistical bank unchanged: no part of the pair is added or removed, and the Last registers are not replaced. Any number being typed is finished before these checks, as with other statistics commands.

A paired observation has an x-value and a y-value. At the moment you press $\Sigma+$, put x in X/S0 and y in Y/S1. The usual key sequence therefore enters *the y-value first*, the reverse of the written pair (x, y):

$$y \quad \textbf{ENTER} \quad x \quad \Sigma+$$

ENTER finishes the first number; starting the second lifts the first into Y. The statistics key finishes the second number and records the pair. For regression, x is the explanatory or independent variable, and y is the response to be fitted.

After addition, X (display S0) contains the new *sample count*, not the entered x. Y and all deeper stack positions stay in place. The original pair is saved in Last X and Last Y. Entry of the next pair can begin immediately; no extra ENTER is needed after $\Sigma+$.

Each successful press adds one pair, including a pair containing zero. It does not read a frequency from Y: that value is the second coordinate, or a weight only when you later request weighted mean. Pressing Σ+ repeatedly is not a way to repeat the original sample, because the first press replaced its x with the count. Re-enter the pair for each additional occurrence.

Single-variable observations

For one-variable work, put the observation in X and use zero for Y. The x-mean and x-standard deviation then describe the intended data; the y-mean and deviation are zero. Enter 0 ENTER before *each* observation to make that convention explicit. Merely clearing the stack once does not ensure that Y stays zero as later digits lift previous counts and results.

Accumulate three measurements

Use the chapter's prepared RPN settings, including real mode, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Enter measurements 8, 10, and 12:

f Σ+ f CLEAR

Continue with	Count in X
0 ENTER 8 Σ+	1.0000
0 ENTER 10 Σ+	2.0000
0 ENTER 12 Σ+	3.0000

STAT is present. Press $\bar{x}$. The mean in X is 10.0000; Y shows 0.0000. The check is $(8 + 10 + 12)/3 = 10$. The measurements remain accumulated after the mean is calculated.

Statistics uses real components even with complex mode on. A pair of complex stack operands does not become four coordinates or one complex observation. Nonzero imaginary data produces the real-only warning in complex mode; use deliberate real inputs for this chapter. New Last operands retain both components only with CMPLX on; with it off, their imaginary components are zero. See sections 6.5 and 8.7 for these distinctions.

10.3 What the six accumulators retain

The calculator retains a count and five sums. It does not maintain a sample list that you can scroll through, select by observation number, or recover in its original order. Keep the original observations when you may need to check, replace, or

recompute them. The totals are enough for the functions in this chapter, but are not a substitute for the source data.

Table 10.2: The statistical accumulators and their ordinary-memory copies.

Accumulator	Copy destination	Meaning
n	R4	Number of pairs added, less pairs removed.
$\sum x$	R5	Sum of first coordinates.
$\sum x^2$	R6	Sum of squared first coordinates.
$\sum y$	R7	Sum of second coordinates, or total weight.
$\sum y^2$	R8	Sum of squared second coordinates.
$\sum xy$	R9	Sum of the products within each pair.

The count n in this table is independent of the financial n register. Entering observations does not change a loan's period count or the cash-flow selector. Nor do the running sums occupy ordinary R4–R9 while data is being entered; those destinations are used only by the explicit copy command below.

Σ+ adds 1 to n, adds x and y to their sums, and adds x^2, y^2, and xy to the remaining sums. Its blue inverse, Σ−, subtracts those same contributions. A negative observation is a valid value: adding -3 contributes -3 to $\sum x$ and 9 to $\sum x^2$. It does not remove a previously entered positive 3.

STAT reports occupied statistical storage, including zeros. Removing the final observation can leave all six values zero while STAT remains present. Only an explicit statistics clear removes that occupancy. Conversely, the flag alone does not establish a positive count or valid data. The bank and its flag survive mode changes and relaunches; see chapters 6 and 14.

10.4 Correct or remove an observation

Before pressing Σ+, use ordinary entry correction if you have mistyped a digit. After a pair has been added, changing the display does not change its contribution to the sums. To correct an accepted observation, re-enter the *incorrect* pair and press g Σ+, selecting Σ−. Then enter the correct pair with unshifted Σ+. The order remains y, ENTER, x for removal as well as addition. Remove both coordinates even if only one was wrong.

Removal returns the reduced count in X, preserves Y and deeper stack positions, and saves the removed pair in the Last registers. The calculator rejects removal when the count is already zero. It cannot check that the pair you subtract was actually entered, because it holds accumulators rather than an observation history. An incorrect subtraction can corrupt the sums without producing an immediate error.

Replace an accepted pair

Use real RPN, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Suppose the intended pairs are $(1,2),(2,4),(3,5),(4,8)$, but the last y is entered as 9. Reproduce the mistake with:

f Σ+ f CLEAR
2 ENTER 1 Σ+ 4 ENTER 2 Σ+
5 ENTER 3 Σ+ 9 ENTER 4 Σ+

The count is `4.0000`. Subtract the wrong pair:

9 ENTER 4 g Σ+

The count becomes `3.0000`. Add the correct pair:

8 ENTER 4 Σ+ $\bar{x}$

After addition the count is 4; after mean, X shows `2.5000` and Y shows `4.7500`. The second-coordinate total is now $2+4+5+8=19$, as intended.

If the incorrect pair was just added and neither Last register has since changed, you can retrieve that exact pair instead of retyping it. Use CLEAR, not CLx: CLx would overwrite Last X with the displayed count. Press f CLEAR, then blue LAST Y, blue LAST X, and blue $\Sigma-$. In physical key locations the sequence is:

f $1/x$ g $1/x$ g $x \leftrightarrow y$ g Σ+

CLEAR discards the working stack but preserves the bank and both saved operands. The recalls put the saved y in Y and the saved x in X, ready for removal. This method recovers the most recently saved operands, not an arbitrary earlier observation; section 2.9 explains that limit.

Adding and then subtracting floating-point contributions need not restore every accumulator bit for bit. For a data set with many corrections, or whenever you cannot identify the original mistake, clear statistics and re-enter the verified source observations.

10.5 Means and weighted means

The unshifted $\bar{x}$ key calculates both arithmetic means:

$$\bar{x} = \frac{\sum x}{n}, \qquad \bar{y} = \frac{\sum y}{n}.$$

There must be at least one observation. The result in X (S0) is $\bar{x}$, and the result in Y (S1) is $\bar{y}$. The calculator lifts both results ahead of the old stack rather than replacing its first two values. On a fixed stack, the two deepest values are therefore lost. Neither mean changes the accumulated data.

Orange $\bar{x}$ calculates the weighted mean of the x-values, treating their paired y-values as weights:

$$\bar{x}_w = \frac{\sum xy}{\sum y}.$$

Enter each weight first, ENTER, then its value, and press $\Sigma+$. The weighted-mean command returns one value by replacing X (S0); it preserves Y and deeper positions. It requires at least one entered pair and a nonzero sum of weights.

Average values with different frequencies

Use real RPN, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Values 10, 20, and 40 occur with frequencies 2, 3, and 1. Enter each frequency as the y-weight:

f $\Sigma+$ f CLEAR
2 ENTER 10 $\Sigma+$
3 ENTER 20 $\Sigma+$
1 ENTER 40 $\Sigma+$ f $\bar{x}$

The weighted mean is `20.0000`, since

$$\frac{2(10) + 3(20) + 1(40)}{2 + 3 + 1} = \frac{120}{6} = 20.$$

Only three pairs were entered, so the statistical count is 3, not 6. Pressing unshifted $\bar{x}$ afterward instead gives `23.3333`, the unweighted average of 10, 20, and 40, with `2.0000`, the mean weight, in Y.

Weights may be fractional or signed; the calculator applies the same quotient. For an ordinary weighted average, choose the nonnegative weights appropriate to the problem. Signed weights can produce a result outside the range of the entered values, and weights that sum to zero are rejected. A zero weight still adds one pair to n and contributes its x to the unweighted sums, even though its contribution to the weighted numerator is zero.

The weighting interpretation belongs only to weighted mean. SD and linear regression continue to treat each entered pair as one equally weighted observation. In particular, SD on the three frequency pairs above is not the deviation of the six expanded observations. To use ordinary sample SD for frequency data, enter each actual occurrence with the single-variable convention, or use an appropriate separate formula.

10.6 Sample standard deviations

At least two observations are required for SD. Press g $\bar{x}$ to select it. It returns the x-sample deviation in X (S0) and the y-sample deviation in Y (S1), lifting two values in the same way as the means. The formulas are

$$s_x = \sqrt{\frac{\sum x^2 - (\sum x)^2/n}{n-1}}, \qquad s_y = \sqrt{\frac{\sum y^2 - (\sum y)^2/n}{n-1}}.$$

The divisor is $n-1$, so these are sample standard deviations, not population standard deviations. Each result has the same units as its observations. Constant observations are permitted and have deviation zero; both coordinates are evaluated by the command.

Find the spread of three measurements

Use real RPN, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Enter the complete single-variable set again:

f $\Sigma+$ f CLEAR
0 ENTER 8 $\Sigma+$ 0 ENTER 10 $\Sigma+$
0 ENTER 12 $\Sigma+$ g $\bar{x}$

The results are `2.0000` in X and `0.0000` in Y. Relative to mean 10, the x-deviations are $-2, 0, 2$, so $s_x = \sqrt{(4+0+4)/(3-1)} = 2$.

If these three values constitute the entire population you intend to describe, its standard deviation instead uses divisor 3. Convert a sample deviation to a population deviation by $\sigma_x = s_x\sqrt{(n-1)/n}$. Show the Scientific panel for the square-root key if needed; changing panels does not disturb the waiting deviation. Continuing directly from the preceding example, with the same settings and no pending prefix, press:

3 ENTER 1 — 3 ÷ $\sqrt{x}$ ×

The result is `1.6330`, from $2\sqrt{2/3}$. This is an explicit calculation, not a setting that changes what SD returns.

10.7 Fit a straight line

The unshifted L.R. key fits the line $\hat{y} = a + bx$ to the accumulated pairs by ordinary least squares in the y direction. It returns slope b in X (S0) and intercept a in Y (S1). The intercept is the fitted value at $x = 0$; the slope is the change in fitted y

per unit change in x. Both results are lifted ahead of the previous stack. L.R. uses the stored pairs, not the two values currently displayed.

At least two pairs are needed, and the x-values must not all be equal. Constant y-values are allowed: their fitted line is horizontal. Reversing the two coordinates fits a different regression; it is generally not the inverse of the original fitted line.

The underlying totals make the calculation explicit. Define

$$M = n\sum x^2 - (\sum x)^2,$$
$$N = n\sum y^2 - (\sum y)^2,$$
$$P = n\sum xy - (\sum x)(\sum y).$$

Then

$$b = \frac{P}{M}, \qquad a = \frac{M\sum y - P\sum x}{nM}.$$

The quantity M must be positive for a usable fitted slope. These are unweighted formulas; the y-coordinate is the response, not a weight as it is for the weighted-mean command.

Fit four paired observations

Use real RPN, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Fit $(x, y) = (1, 2), (2, 4), (3, 5), (4, 8)$:

f Σ+ f CLEAR
2 ENTER 1 Σ+ 4 ENTER 2 Σ+
5 ENTER 3 Σ+ 8 ENTER 4 Σ+ L.R.

The slope in X is `1.9000`; the intercept in Y is `0.0000`. The fitted line is $\hat{y} = 1.9x$. The totals give $M = 20$, $P = 38$, $\sum x = 10$, and $\sum y = 19$, so the slope is $38/20$ and the intercept is $(20 \times 19 - 38 \times 10)/(4 \times 20) = 0$.

The same stored data can supply means and deviations without re-entry. After the fit above, unshifted $\bar{x}$ returns `2.5000` in X and `4.7500` in Y. Blue $\bar{x}$ then returns `1.2910` and `2.5000`, respectively. Each request pushes another pair of results; inspect the stack or store useful values before requesting more results on a small fixed stack.

10.8 Estimate a response and read correlation

The combined estimate-and-correlation operation requires at least two observations and positive variance in both coordinates. If all y-values are identical, L.R. can still return the horizontal line, but the estimate key rejects the combined result because correlation is undefined. Use the line's constant intercept for its fitted response in

that case. There is no separate key that estimates x from y; choose and enter the explanatory coordinate before fitting the model you need.

To estimate y for a chosen x, put that new x in X and press f L.R., selecting the orange $\hat{y}$ legend. You do not need to press unshifted L.R. first. The command calculates from the current accumulators, consumes the input x, and returns two values: the fitted $\hat{y}$ in X (S0) and correlation r in Y (S1). The old Y moves to S2. It does not add the requested point to the data.

Correlation is

$$r = \frac{P}{\sqrt{MN}},$$

with M, N, P defined in section 10.7. For valid finite data, its mathematical range is -1 through 1. A positive value indicates that larger x tends to accompany larger y; a negative value indicates the opposite direction. Perfect increasing and decreasing straight lines have $r = 1$ and $r = -1$. A value near zero indicates little linear association, but does not rule out a nonlinear relationship.

Interpolate and extrapolate from the fitted data

Use the exact four-pair data set and RPN settings prepared in section 10.7; if necessary, repeat its complete entry sequence. The bank must contain just those four pairs. With no pending shift or command, press:

f CLEAR 2.5 f L.R.

The estimated response is `4.7500` in X, and correlation is `0.9812` in Y. This evaluates 1.9×2.5 within the observed range of x. For a second request, press:

f CLEAR 5 f L.R.

The estimate is `9.5000`, with the same correlation `0.9812`. The value 5 lies beyond the observed range 1 through 4, so this second request extrapolates the fitted line. Neither request changes the count of four observations.

The example's correlation is $38/\sqrt{1500}$, approximately 0.9811557810. Correlation belongs to the stored data and therefore does not depend on the new x supplied for an estimate. The function does not provide a confidence interval, prediction interval, or a test that a straight line is suitable. A high correlation also does not make extrapolation reliable or establish a causal relationship.

10.9 Inspect the totals and preserve useful results

RCL_{Σ} copies all six accumulators into ordinary R4–R9 in the order shown in table 10.2. This *overwrites* those six registers. Preserve anything needed from them before copying, then press g L.R. to select the command. Ordinary R0–R3 and

the statistical bank itself are unchanged. Like any memory write, this copy includes both components with CMPLX on and only the real component with CMPLX off; it never strips a component from the source statistical bank.

The copy finishes an active numeric entry, but does not move stack values or change either Last register. It does not place six totals directly on the stack. Use ordinary RCL and a digit to read an individual copied value. Those later recalls have the ordinary lift and Last X effects described in chapter 6.

Check all six totals for the four-pair data set

Use real RPN, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Prepare exactly the four pairs with the complete sequence in section 10.7. This example deliberately replaces R4–R9. Press g L.R., then use each recall below to inspect the copied total:

Keys	Total	Display
RCL 4	n	4.0000
RCL 5	$\sum x$	10.0000
RCL 6	$\sum x^2$	30.0000
RCL 7	$\sum y$	19.0000
RCL 8	$\sum y^2$	109.0000
RCL 9	$\sum xy$	57.0000

Each recall displays the indicated value while preserving its ordinary memory copy. **MEM** is present as well as **STAT**. For example, the final total is $1(2) + 2(4) + 3(5) + 4(8) = 57$.

These are snapshots of the totals at the time of copying. Adding or removing a sample afterward does not refresh them; press RCL_{Σ} again when you want current copies. Editing R4–R9 likewise does not edit the statistical bank. There is no inverse keyboard command that loads these six ordinary memories back into the accumulators.

Copying an empty statistical bank stores six zeros and lights **MEM**, but does not light **STAT**. Clearing statistics later leaves the copies intact. In ALG, Recall summations both displays the six labeled totals and copies their real components to R4–R9. Ordinary RCL can then use those copies in an expression. Unlike RPN's copy command, the ALG worksheet projects the totals to real values even with CMPLX on; the source statistical bank is unchanged.

10.10 Statistics worksheets in ALG

Select ALG and press orange FUNC for STAT. Choose the required operation in the worksheet. The main expression remains suspended; all fields, result selections, and unfinished edits are saved. The field controls follow section 4.12: equals evaluates a field, ENTER accepts it, and the named action performs the operation. Successful equals and ENTER also update ANS with the field value; neither operation adds or removes a sample by itself. On the first visit, the sample's X field starts from a valid displayed operand and Y starts at zero. Later visits retain the accepted coordinates and any unfinished edits, so check both fields when starting another data set.

Table 10.3: ALG statistical inputs and results.

Operation	Inputs	Outputs
Add or remove sample	X and Y (use Y = 0 for one-variable data)	Updated sample count.
Means	Existing bank	Mean X and Mean Y.
Weighted mean	Existing bank; Y supplies weights	Weighted mean.
Sample standard deviations	Existing bank	Standard deviation X and Y.
Linear regression	Existing bank	Slope, Intercept, and Correlation when defined.
Estimate Y	X to estimate; existing bank with nonzero X and Y variance	Estimated Y and Correlation.
Recall summations	Existing bank	Count and all five sums, also copied into R4–R9.
Clear statistics	None	Cleared count; removes the six accumulators.

Accepting one coordinate does not add or remove a sample. Both finite coordinates, the existing totals, and every updated total must validate before ADD SAMPLE or REMOVE SAMPLE writes the bank. A failed sample operation leaves all six accumulators unchanged. Removal uses the same X and Y as the original sample; it cannot verify that those coordinates were actually present, because individual samples are not retained.

Each successful calculation updates ANS with its declared primary output: the sample count, Mean X, weighted mean, Standard deviation X, Slope, or Estimated Y, respectively. Recall summations and Clear statistics use the count as their primary output too. Selecting another output does not change ANS. Use value inserts the selected result into the main expression. For a constant Y series, a valid regression can have slope zero and a finite intercept while correlation is undefined; ALG keeps the valid outputs and omits correlation.

To enter the pair $(X, Y) = (2, 5)$, choose the add-sample worksheet, accept X as 2 and Y as 5, then ADD SAMPLE. Repeating ADD SAMPLE deliberately adds another copy of that pair. To correct it, choose remove sample and accept the original 2 and 5 before REMOVE SAMPLE, then add the corrected pair. To abandon an unaccepted coordinate edit, use CANCEL; previously accepted fields and completed sample operations remain.

10.11 Stack effects, errors, and numerical limits

The RPN functions share one data bank but have different stack effects. Table 10.4 summarizes successful commands after any active numeric entry has finished. A pair result is ordered with the first value in X (S0) and the second in Y (S1). On an unlimited stack, two-result queries add two positions; estimation consumes one input and adds two results, for a net increase of one. On a fixed stack, each lift discards a deepest value as usual. Where a command saves old Y, it does so only if that stack position exists; with only one unlimited-stack operand, Last Y is preserved.

Table 10.4: Statistical results and saved operands.

Command	Stack effect	Last registers
$\Sigma+$, $\Sigma-$	Replace X with count; leave Y and deeper values in place.	Save old X and Y.
Means	Lift $\bar{x}$ in X, $\bar{y}$ in Y; old X moves to S2.	Save old X and Y.
Weighted mean	Replace X; leave deeper values in place.	Save old X; preserve Last Y.
SD	Lift s_x in X, s_y in Y; old X moves to S2.	Save old X and Y.
L.R.	Lift slope in X, intercept in Y; old X moves to S2.	Save old X and Y.
Estimate	Consume input X; return $\hat{y}$ in X, r in Y; old Y moves to S2.	Save input X; preserve Last Y.
Clear or copy sums	No movement of committed stack values.	Preserve both.

The Last registers contain the old working operands, not a backup of the six accumulators. A mean, SD, or L.R. request can therefore replace the pair saved by the most recent sample addition. Use the explicit re-entry correction method if those saved operands have changed.

A statistics-domain failure leaves the accumulated bank and committed stack values unchanged. A number being typed may already have been accepted before the error was found. Check **ENTRY** before using backspace; it may now act as CLx instead of deleting a digit. Correct the input or data and retry.

Table 10.5: Conditions to check after a statistics error.

Operation or message	Check
Statistics samples must be finite	Replace NaN or infinity in the real component of X or Y. The rejected pair has not changed the bank. Imaginary components are not sample coordinates.
Statistics totals must be finite	An existing total is nonfinite, or the proposed update overflows. All six totals are retained. Clear and re-enter data if the existing bank is nonfinite.
Removal from empty data	At least one pair must remain before $\Sigma-$. A positive count still cannot prove that the particular removed pair was entered.
Mean or weighted mean with no data	Enter at least one pair.
Weights sum to zero	Inspect the entered y-weights and their signs; their sum must be nonzero.
SD, regression, or estimation needs two points	Enter at least two pairs, rather than one pair with a large weight.
X has zero variance	The x-coordinates cannot all be equal for a regression or estimate.
Y has zero variance	Correlation needs varying y, even though a horizontal fitted line can be calculated by L.R.
Negative variance	Check for an unmatched removal, damaged totals, or cancellation from insufficient working precision.

Large offsets and small variation

Statistics calculates with the current working precision; changing the display does not improve the stored sums. To recompute at a higher precision, select it first, clear statistics, then re-enter the original observations. Increasing precision cannot recover detail already lost from the accumulators; see chapter 5.

The variance and regression formulas subtract large, potentially nearly equal quantities such as $n\sum x^2$ and $(\sum x)^2$. When observations have a large common offset but differ only slightly, those terms can lose the small difference that contains the variance. The current calculation uses these raw sums; it is not a centered streaming variance algorithm. Cancellation can produce an inaccurate small variance, zero, or a negative value that triggers an error. More displayed digits alone cannot distinguish these cases.

Choose adequate working precision before accumulation, and compare a sensitive result after recomputing from the original data at a higher setting. Another useful mathematical approach is to subtract a known common offset from the observations before entering them. Subtracting a constant leaves standard deviation unchanged; add the offset back to the calculated mean. For regression, keep track of any shifts in both coordinates when interpreting the intercept and supplying a new point. Chapter 5 explains cancellation and retained values in more detail.

The finite-value checks protect sample addition and removal; they do not establish that the data or the resulting statistical interpretation is correct. A saved bank already containing nonfinite totals cannot accept another addition or removal. ALG additionally rejects nonfinite stored totals before any statistical query, and rejects nonfinite computed outputs. Its Clear statistics action remains available even for a contaminated bank. Clear such a bank and re-enter verified observations. If finite totals cannot be trusted, check the simple causes first: swapped coordinates, a forgotten statistics clear, an incorrect pair removed with Σ−, weights mistaken for counts, or a sample deviation mistaken for a population deviation.

Chapter 11

Time Value of Money, Loans, and Savings

Time value of money calculations relate an amount now, an amount later, and equal payments between them. Supply the known quantities and solve for the missing one: a loan payment, an affordable principal, a savings balance, an interest rate, or a number of periods. The financial registers retain the inputs, so you can change one assumption and calculate again.

These operations are available in both entry modes. Show the Financial panel by swiping left across the top two keyboard rows in portrait; both panels are visible in a wide landscape layout. The register meanings, cash signs, period units, payment timing, and formulas below apply in both modes. RPN readers should follow section 11.4 and the worked key sequences. ALG readers should start with section 11.12 for explicit stores, CPT, and the amortization worksheet, then use the problem sections for their assumptions and interpretation.

In the RPN sequences, X means the current stack position labeled S0 in the full-stack view; Y means S1. Last X and Last Y hold separate saved operands. The financial registers and ALG worksheet fields are separate from these stack positions; see chapter 6.

11.1 The five financial registers

The main keys n, i, PMT, PV, and FV identify five persistent financial registers. They are separate from the RPN stack and ordinary memories. An input stays in its register when you clear the stack, perform unrelated arithmetic, switch entry modes, or close and reopen the application.

A solve uses *all four other registers*. For example, solving PMT uses n, i, PV, and FV, even when the final balance was stored much earlier than the other inputs. The old value of the register being solved is irrelevant and is replaced by the answer. The order or age of the input stores does not choose which terms participate.

A cleared or never-written TVM register supplies zero. You need not have explicitly stored four entries for a solve to run, but each of the four quantities must have the intended value. Explicitly setting a zero payment or zero final balance makes a reusable sequence easier to check. Old nonzero values remain active until replaced or cleared.

Table 11.1: The TVM quantities. A period is the common interval chosen for the rate and the payments.

Register	Meaning
n	Number of payment or compounding periods.
i	Interest percentage per period; enter 0.5 for half a percent per period.
PV	Present value: the lump sum at time zero.
PMT	Equal payment each period. Use zero when there are no periodic payments.
FV	Future value: the final lump sum or residual at time n. Use zero for a loan fully paid by its regular payments.

The **FIN** flag announces at least one occupied financial register, including a stored zero. It does not certify complete or valid TVM data. An unset register recalls as zero without becoming occupied. The registers are also used by cash-flow and other financial commands: for example, cash-flow entry changes n. Re-establish the intended TVM inputs after using those tools; see chapters 12 and 13.

All participating values and results must be finite. The periodic rate must be greater than −100 percent. Entered n must be nonnegative; solving i or PMT requires positive n, while PV and FV permit $n = 0$. A solved n cannot be negative. At zero interest the simpler balance equation applies, but some inputs still do not determine a unique result. For example, with zero PMT and exactly opposite PV and FV, every n balances at zero interest; there is no unique period count to return.

11.2 Cash signs, periods, and interest rates

Choose one viewpoint and use it consistently. Money received is positive; money paid is negative. A borrower normally enters a positive PV for the loan received, negative PMT for repayments, and a negative FV if an additional balloon payment remains. A saver enters a negative initial deposit and negative regular deposits, with a positive future withdrawal. Reversing every cash amount's sign reverses the monetary results without changing the rate or number of periods.

Use +/− to enter a negative amount. The subtraction key is an arithmetic operation, not a sign-entry key. Currency symbols and units are not entered or stored; express every cash amount in the same currency and scale. These calculations contain no separate fees, taxes, or changing-rate schedule unless you include their effects in a model appropriate to the problem.

The rate in i is a *percentage per payment period*, and n counts those same periods. Monthly payments require a monthly rate and a count of months. Annual payments require an annual rate and a count of years. The calculator has no separate payments-per-year setting that silently converts either input.

Table 11.2: Matching a nominal annual rate to its stated compounding and payment interval.

Schedule	n	i
5 years, annual periods at 6% annually	5	6
5 years, monthly periods at nominal 6%, compounded monthly	60	0.5
5 years, quarterly periods at nominal 6%, compounded quarterly	20	1.5

The two annual-to-monthly shortcuts

Orange n carries 12×. Enter years, then press f n; the calculator multiplies X by 12, replaces X with the product, and stores it in n. Orange i carries 12÷. Enter a nominal annual percentage, then press f i; the quotient by 12 replaces X and is stored in i. Neither shortcut changes deeper stack positions or either Last register. Each prepares the next bare TVM key to solve.

These are exactly multiplication and division by 12. They do not choose a calendar or recognize a quoted rate's convention. An *effective* annual percentage j, for example, is not converted to an equivalent monthly percentage by simply dividing by 12. For m equal periods per year, the equivalent periodic percentage is

$$i = 100\left[\left(1+\frac{j}{100}\right)^{1/m}-1\right].$$

Conversely, a periodic percentage i has effective annual percentage $100[(1+i/100)^m-1]$. Identify the rate convention before choosing the arithmetic. The shortcuts suit the nominal monthly rate convention used by the loan examples below.

11.3 END and BEGIN payment timing

In END mode, the equal payments occur at the end of each period: at times 1 through n for an integer n. In BEGIN mode they occur at times 0 through $n-1$. PV is at time zero and FV is at time n in either case. BEGIN therefore moves every payment one period earlier; it does not move either lump sum.

BGN visible means BEGIN; its absence means END. Check the flag before toggling, because the same sequence can select either state. To toggle payment timing, press g n, the blue BGN function. There is no separate END key. The choice survives clearing the financial registers and survives relaunch.

For a positive rate, an earlier savings deposit earns an extra period of interest. An earlier loan payment likewise needs a smaller magnitude to have the same present value. With zero payments, timing has no effect; with zero interest, the distinction also disappears.

Let $r = i/100$, $q = 1+r$, and let $b = 0$ for END or $b = 1$ for BEGIN. The relationship being solved is

$$PV\,q^n + PMT\,q^b\frac{q^n - 1}{r} + FV = 0. \tag{11.1}$$

At zero interest its continuous limit is $PV + n\,PMT + FV = 0$. This explains why both the periodic payment and final balance can participate in the same calculation. For fractional n, see section 11.10 before interpreting the result as a payment schedule.

In RPN, changing BEGIN/END preserves whether the next TVM key is ready to store or solve. After a successful PMT calculation, for example, you may toggle timing and press PMT again to recompute from the retained inputs. You do not need to re-enter them merely to compare the two timings. In ALG, use CPT followed by PMT for each new calculation; its bare PMT key always stores.

11.4 When an RPN financial key stores and when it solves

A *bare* TVM key means its unshifted main function, without STO or RCL before it. After number entry, ordinary arithmetic, or a recall, the next bare TVM key stores the current X in that register. After a financial-register store or a TVM solve, the next bare TVM key solves for its variable. The two monthly shortcuts also leave the calculator ready to solve.

This makes a typical sequence read naturally: enter each known number and press its financial key, then press the unknown's key without entering another number. Each fresh numeric entry selects storage for the following key. The final store selects calculation for the bare unknown key that follows it.

Do not insert ENTER between the final known-value store and the unknown key. ENTER is an ordinary stack action; the following financial key would store instead of solve. Likewise, a recall used to inspect the inputs changes the next bare key to storage. There is no separate store/solve readiness annunciator: **FIN** reports bank occupancy, not the meaning of the next key.

Explicit storage and recall

STO PV always copies X to PV; RCL PV always recalls PV. The other four main keys work the same way as targets. These prefixes override the bare key's store/solve choice. Financial STO finishes any active numeric entry and preserves the committed stack and both Last registers. RCL finishes an entry, lifts the recalled value into X, and preserves both Last registers. The source register is unchanged.

Arithmetic storage is available too. For example, STO + PV adds X to the existing PV; STO − PV subtracts X from PV. The current X stays in place,

Table 11.3: The next bare TVM key after common actions.

Preceding action	Next bare n, i, PV, PMT, or FV
Type a number; perform ordinary arithmetic; recall a value	Store X in the selected register.
Store in a financial register, including explicit or arithmetic STO	Solve the selected variable.
Complete a TVM solve or use 12×/12÷	Solve the selected variable.
Press ENTER, CLx, CLEAR, or an ordinary stack command	Store X in the selected register.
Toggle BEGIN/END	Keep the preceding store/solve readiness.
Use AMORT, RND, cash-flow, date, bond, or depreciation commands	Store X in the selected register.

and the next bare TVM key solves. See section 6.3 for all four modifiers and operand order.

If you have inspected inputs and want an unambiguous solve, first check the four actual inputs. Recall and explicitly restore any *known input*, then press the target key. For example, f CLEAR n STO n PMT retains the known n and solves PMT. This avoids depending on the action that happened before the sequence.

A successful solve both stores its answer in the target register and lifts that answer into X (display S0). The previous X moves to Y (display S1); deeper values move as for an ordinary lift. Both Last registers are preserved. The next bare TVM key can solve again immediately. By contrast, a financial store leaves X in place rather than lifting a second copy.

11.5 Clearing and preparing a financial problem

f CLEAR, the orange function of $1/x$, clears the RPN stack and unfinished entry while preserving financial inputs. Orange FV, labeled CL_{fin}, removes the five financial-register entries but leaves cash flows and payment timing intact. Use the financial MCLR group if both banks should be discarded; section 6.10 gives its sequence and explains how each clear handles an unfinished entry.

The prepared TVM start used below

Use RPN, real mode (**COMPLEX** absent), a fixed stack of 5, 600 working digits, and FIX 4. Set these by the procedures in sections 3.1 and 5.2. Select the Financial panel. These examples deliberately replace the TVM inputs; preserve any financial values you still need before starting them. With no open tool, active shift, or pending command, press f CLEAR, then f FV for CLfin. Check that **BGN** is absent; if present, press g n once to select END.

Each independent worked example below repeats the stack and financial clears and states any change from these settings. Ordinary memories, statistical data, and the cash-flow list need not be empty. These examples are unaffected by angle units and date order.

11.6 Loan payments and a final balloon

For a level-payment loan, store the number of payments in n, periodic percentage in i, amount received in PV, and desired final payoff in FV. Then solve PMT. Use FV equal to zero when the regular payments should pay off the entire balance. A negative FV instead describes an additional amount the borrower pays at the final date.

Solve the monthly payment on a loan

Use the prepared TVM settings: RPN, real mode, fixed depth 5, 600 working digits, FIX 4, END, and no pending prefix. A loan of 100 000 is repaid monthly over 30 years at nominal annual 6 percent, compounded monthly, with no balloon. Press these rows in order:

Keys	Meaning or display
f CLEAR f FV	Clear stack and TVM bank.
30 f n	Store n = 360.
6 f i	Store i = 0.5.
100000 PV	Store the amount received.
0 FV	No final balloon.
PMT	-599.5505

The negative result is a monthly payment by the borrower. Its retained value begins −599.550525152752 ...; FIX 4 has not rounded the stored PMT to cents. There is no ENTER before the final PMT key.

After this result, changing a known value and solving again requires only that new input and the target key. For example, with the same loan still stored and

END selected, 15 f n stores 180 months; PMT then recalculates the payment. Re-enter the complete preparation when you want an independent problem.

Leave a balloon balance at the final date

Use the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings, with no pending command. A borrower receives 18 000, makes 36 monthly payments at 0.625 percent per month, and pays an additional 4 500 at the end. Enter:

f CLEAR f FV
36 n .625 i 18000 PV
4500 +/− FV PMT

The result is -448.0589. The final date contains both the last regular payment and the balloon. The FV input is negative because that final lump sum also leaves the borrower.

To find the residual for a chosen payment instead, set n, i, PV, and PMT, then solve FV. The sign describes the final cash flow required to balance the model. A nonzero final result does not disappear because the chosen payment was displayed to two decimal places.

11.7 Present value and a payment budget

Solving PV discounts the periodic payments and final amount to time zero. It can answer how much principal a payment budget supports, or the present value of an annuity. Keep the same sign convention for payments and any terminal amount; the calculator supplies the balancing present-value sign.

Find the principal supported by a monthly budget

Use the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings, with no pending prefix. A monthly payment of 750 is available for 60 months at 0.5 percent per month, with no final balance. Press:

f CLEAR f FV
60 n .5 i
750 +/− PMT 0 FV PV

The result is 38,794.1706. The positive PV is the amount received now in exchange for the specified outgoing payments. It excludes any separate costs not represented by those cash flows.

For an annuity that *pays you* 750 each month, enter PMT as positive instead. With the other inputs unchanged, the balancing PV is negative: an amount paid now to

receive that stream. BEGIN timing would value the same number of payments one period earlier.

11.8 Future value and regular savings

To accumulate an initial deposit and regular additions, enter the initial deposit as negative PV and each addition as negative PMT. Solve FV for the final positive withdrawal. Both deposits participate; setting PMT does not replace or suppress PV. For compound growth without additions, explicitly set PMT to zero.

BEGIN savings includes the first PMT at time zero *in addition to* PV. Do not include the same deposit in both quantities unless two separate deposits actually occur then. For unequal deposits or irregular amounts at equally spaced periods, use the cash-flow procedures in chapter 12 rather than a single PMT.

Accumulate an initial deposit and monthly additions

Use the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings, with no pending prefix. Deposit 1 000 now and 200 at the end of each of 60 months, at 0.5 percent per month. Press:

f CLEAR f FV
60 n .5 i
1000 +/− PV 200 +/− PMT FV

The result is `15,302.8563`. Now compare deposits at the beginning of each month, keeping the same initial deposit:

g n FV

The **BGN** flag appears, and the new result is `15,372.6263`. Each periodic deposit earns one extra month of interest; the initial 1 000 is unchanged. Press g n again to return to END before another example.

11.9 Solving for the periodic interest rate

With periodic payments, i generally requires a numerical solve. The solver searches for a rate greater than −100 percent per period; negative rates within that domain are permitted. It uses all of PV, PMT, and FV, so an old final balance can change the answer. The old i is not an input or an initial guess for its own TVM solve.

A valid rate requires a positive n and cash flows that can balance. All nonzero amounts with the same sign cannot do so. Some opposite-sign combinations are still impossible or indeterminate. Unusual combinations can also lack a uniquely

meaningful rate; the command returns one solution when its search succeeds, not a catalog of every possible root. The bounded IRR tool belongs to cash-flow calculations, not this TVM i key.

To solve i, set n and all three cash quantities, then press i directly after the final financial store. The answer is a percentage for the period represented by n. A five-year calculation with annual periods returns an annual percentage; a calculation counted in months returns a monthly percentage. No annualization is applied automatically.

Find the annual rate for a lump sum's growth

Use the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings, with no pending command. An initial payment of 1 000 grows to a receipt of 1 500 after five annual periods, with no intervening payments. Enter:

f CLEAR f FV
5 n 1000 +/− PV
0 PMT 1500 FV i

The result is `8.4472` percent per year. Independently, $100[(1500/1000)^{1/5}-1] = 8.447177\ldots$. Timing is immaterial here because PMT is zero.

11.10 Solving n and interpreting fractional periods

To solve n, set i, PV, PMT, and FV. The calculator first obtains the period count implied by those inputs, then stores and displays a whole number. The nominal cutoff is 0.005: a smaller fractional part rounds down to the integer part, while a part at or above the cutoff rounds up. A small numerical guard also rounds values just below the cutoff upward; the exact rule is in section C.6. This special TVM rule is independent of FIX and differs from ordinary nearest-integer rounding.

Count whole periods needed for a lump sum to double

Use the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings, with no pending command. Find the period count for 1 000 to become 2 000 at 5 percent per period, without regular payments:

f CLEAR f FV
5 i 1000 +/− PV
0 PMT 2000 FV n

The result is `15.0000`, and n stores 15. The unrounded mathematical count is $\ln 2 / \ln 1.05 = 14.206699\ldots$, whose fractional part requires rounding up. If a period is a year, the result means 15 whole years.

The rounded n does not promise that every other quantity balances exactly at that integer. After solving n, you can solve PMT again for equal payments over that whole number of periods, or solve FV to inspect the resulting residual. Do not interpret a rounded-up loan count as instructions to pay the full regular amount once more without checking the final balance.

To check the cutoff directly, use the prepared TVM settings with no pending prefix. For each row in table 11.4, press:

f CLEAR f FV 0 i listed PV PV
100 +/− PMT 0 FV n

The last column shows the stored whole count, without FIX 4's trailing decimal zeros.

Table 11.4: The 0.005 cutoff at zero interest.

PV	Mathematical $-PV/PMT$	Solved n
200.49	2.0049	2
200.50	2.0050	3
250.00	2.5000	3

Manually entered fractional n

Typing a fractional n and storing it does *not* apply this rounding rule. Other TVM solves retain that value and evaluate the continuous compound-annuity formula in equation (11.1). This does not schedule a literal fraction of an equal payment. It is a mathematical convention that you must match to the intended transaction.

For example, with the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings and no pending prefix, press:

f CLEAR f FV 2.5 n 5 i

1000 +/− PV 0 PMT FV

The display is `1,129.7263`, from $1000(1.05)^{2.5}$, and the stored n remains 2.5. SpicyRPN does not implement the HP initial odd-period simple-interest convention or its optional compound-interest C setting. Do not use a fractional n as an assumed substitute for those conventions or for date-based accrual.

11.11 Amortization: interest, principal, and balance

AMORT, the orange function of PMT, follows a balance through a specified number of equal payments. It returns the total interest and principal portions for that block and updates the balance. Its use of n differs from a TVM payment solve: here n counts payments *already processed*, not the original contractual term.

AMORT requires nonzero PV and PMT with opposite signs, a finite rate greater than −100 percent, and a nonnegative integer stored n. The requested count must be from 0 through 100 000. Both the requested count and the elapsed count must be whole numbers. Count zero returns zero interest and principal and leaves the balance and elapsed count numerically unchanged; all input validation still applies. An unlimited stack must contain at least X and Y because the command saves both operands, although Y is not a financial input.

There is no automatic stop at a contractual term, at a specified FV, or at payoff. A large count can carry a balance past zero, and an insufficient payment can increase its magnitude. Choose the block deliberately and check the updated balance before interpreting another full payment as appropriate.

Before the first block, store the starting signed balance in PV, the periodic percentage in i, the signed payment in PMT, and zero in n. In RPN, all four registers must actually contain entries for AMORT; unlike a TVM solve, this command does not supply missing required registers as zero. FV is not used. Set BEGIN or END as intended. Enter the number of additional payments in X, then press f PMT.

For an ordinary borrower in END mode, PV is positive and PMT is negative. For each payment the calculation is

$$\text{interest portion} = -PV\frac{i}{100}, \qquad \text{principal portion} = PMT - \text{interest portion}.$$

The principal portion is added to PV to get the next signed balance, and n increases by one. In BEGIN mode, interest is zero only for the first payment when the starting

n is zero. Later payments accrue interest normally, including the first payment of a later block whose starting n is already positive.

Table 11.5: AMORT inputs, outputs, and changes.

Location	Meaning
Input X (S0)	Additional whole payments to process, from 0 through 100 000.
Output X (S0)	Total signed interest portion for this block.
Output Y (S1)	Total signed principal portion for this block.
Output S2	The value that was in Y before AMORT, if the stack has room for this third position.
Financial PV	Updated signed balance after the block.
Financial n	Previous elapsed count plus the requested count.
Last X, Last Y	The requested count and the previous Y, respectively.

AMORT consumes the count in X and inserts two results, so the old Y moves to S2 when that position exists. A two-register fixed stack retains only the two results; its previous Y is available in Last Y. Interest is the main result, not the remaining balance. Use the full-stack inspector to read principal without disturbing the outputs, or use $x \leftrightarrow y$ to bring it into X. Recall **PV** for the balance and **n** for the cumulative count. These recalls lift values, so they change the positions of the earlier interest and principal outputs.

Separate two payments, then continue for one more

Use the prepared RPN, real, depth-5, 600-digit, FIX 4, END settings, with no pending prefix. Explicitly initialize the four AMORT registers:

f CLEAR f FV
0 STO n 1 STO i 1000 STO PV
100 +/− STO PMT

Clear only the stack, put a marker in Y, and process two payments:

f CLEAR 77 ENTER 2 f PMT

The outputs are X = `-19.1000` interest and Y = `-180.9000` principal; S2 still contains 77. The financial registers now hold n = 2 and PV = 819.1. The two payments total −200, equal to the sum of the two output portions.
Continue without clearing the financial bank:

1 f PMT

This processes only the next payment. X is `-8.1910`, Y is `-91.8090`, and the registers become n = 3 and PV = 727.291. To verify them, press RCL PV, then RCL n; the respective readouts are `727.2910` and `3.0000`.

The arithmetic of the first block can be checked directly:

Payment	Opening balance	Interest	Principal	Closing balance
1	1000	−10	−90	910
2	910	−9.1	−90.9	819.1

For a BEGIN comparison, repeat the example from the initial clears, including explicit n = 0 and the other three stores. Before processing the count 2, press g n to show BGN. The two-payment totals are interest −9, principal −191, and balance 809; n becomes 2. A following count 1 gives interest −8.09, principal −91.91, balance 717.09, and n = 3. The first payment of that second block does not receive another zero-interest exemption. Toggle BGN off afterward if you want END again.

Continuing or restarting amortization

For successive blocks, retain the updated PV and n and enter only the next block's count. For a repeat from the beginning, restore the original PV and n = 0. Do not leave the original loan term in n after solving PMT: AMORT would treat it as payments already processed. Clearing the stack between blocks is harmless; clearing the financial bank removes the inputs needed to continue.

11.12 TVM and amortization in ALG

Enter a financial input, then press its unshifted register key to store it. A bare key always stores, regardless of preceding financial work. For example, `360` **n** stores 360 periods. **RCL** **n** recalls that input; **CPT** **n** solves for it. Explicit STO and arithmetic STO also accept financial register targets. A store uses the displayed operand and preserves a pending expression: $200 + 50$, followed by PV, stores 50. Evaluate with equals first when you intend to store the total.

CPT requires an unshifted target. CANCEL dismisses it without changing inputs. A pending compound expression produces `Finish the expression first`; complete it with equals or clear it before solving. A lone entered scalar is allowed. A saved CPT prefix resumes as a prompt after relaunch, not as a computation. The orange $12\times$ and $12\div$ shortcuts and blue RND are direct calculations and also require the main expression to be finished. The shortcuts store their result in n or i respectively; RND changes the working result only. All three display a completed result and update ANS. BEGIN/END and the financial clear commands retain their usual shared-bank meanings while preserving the ALG expression.

To solve a loan payment in ALG, clear or complete the main expression, set END, then store each input: n = 360, i = 0.5, PV = 200000, and FV = 0. Press CPT followed by PMT. The result is approximately −1199.1010503 per month. It is stored in PMT and displayed as a completed result; ANS receives the same value. Starting a number begins new work; an operator continues from the payment. This example uses a 6 percent nominal annual rate divided by 12, and its cash-flow signs follow the convention in section 11.2.

Orange PMT opens the Amortization worksheet. Its inputs are Payments to apply, Payments already applied (n), Interest per period % (i), Current balance (PV), and Payment (PMT). This is a count of additional payments applied to the current balance, not arbitrary first/last payment-number bounds. Mapped fields initially read the shared financial registers; an unset register appears as zero. Review those values explicitly. This differs from RPN AMORT, which requires all four financial inputs to have stored entries. ENTER on a mapped field stores its financial register. After accepting all edits, COMPUTE returns Interest, Principal, Balance, and the new count already applied. It updates PV and n together and makes Interest the primary ANS result. BEGIN/END controls timing as in RPN. After a successful computation, the worksheet refreshes n and PV from the updated bank. Pressing COMPUTE again therefore applies the same payment count to the next block, rather than repeating the original block.

Accepting a financial field is a deliberate bank change. Canceling later edits does not undo accepted registers or completed amortization. After computation, check the updated n and PV before applying another payment count. The worksheet keeps the main expression unchanged; Use value transfers one selected output when

you want it in that expression. See section 4.12 for field entry, cancellation, and relaunch behavior.

11.13 Money rounding, precision, and practical limits

TVM solves use the retained input values, not the digits visible under FIX. AMORT also retains its intermediate precision: it does not round each interest portion, principal portion, or running balance to the display count. Changing FIX from four places to two therefore changes the readout without converting a computed payment or schedule to cents. This distinction matters when comparing results with a schedule that rounds individual postings.

To deliberately round a payment, recall it into X, select the desired display count, apply **RND**, and store the rounded result back in PMT. RND is blue PMT. It rounds X to that many decimal places with exact halfway cases away from zero, saves the previous X in Last X, and leaves deeper stack positions and Last Y unchanged. It does not automatically change the source financial register. SCI and ENG use the same count of decimal places for RND, not the number's displayed significant-digit scale; FIX makes the intended money rounding easier to see. See section 5.6.

Use a cent-rounded payment for one year's amortization

Repeat the complete first loan example in section 11.6: RPN, real mode, fixed depth 5, 600 working digits, END, no pending prefix, n = 360, i = 0.5, PV = 100 000, FV = 0, and a freshly solved PMT. Its retained value is −599.550525 .... Now press:

2 **f** **7** **RCL** **PMT** **g** **PMT** **STO** **PMT**

The orange 7 selects FIX 2 and consumes the count. RND then rounds the recalled payment, and STO saves the working-precision representation of −599.55 in financial PMT. Reset elapsed n and process twelve payments:

0 **STO** **n** 12 **f** **PMT**

X displays interest `-5,966.59`; Y displays principal `-1,228.01`. **RCL** **PV** shows the balance `98,771.99`, and **RCL** **n** shows `12.00`. The internal balance begins 98771.9947657778: AMORT rounded neither its monthly portions nor that retained balance to cents. The two-place displays are rounded presentations of the totals.

Rounding only the payment is not the same as rounding every monthly posting. A transaction whose rules round interest and balance at each step requires those rules to be applied explicitly; AMORT has no per-payment rounding setting. Retain

the original unrounded TVM payment separately if you need to compare the ideal equal-payment model with the rounded payment actually used.

Numerical limits and failed solves

The iterative interest calculation uses a default numerical stopping tolerance of 10^{-30}, with bounded search and iteration. Selecting 600 working digits does not promise 600 correct digits in a solved interest rate. Extreme scales, cancellation, or a poorly determined cash-flow pattern also affect reliability. See section C.9 for the stopping rule and chapter 5 for recomputing at higher precision.

A failed TVM solve leaves the target register and the already committed stack values unchanged. An active number can have been committed before the error is detected, so inspect the entry state before deleting digits. Correct the actual input, store it explicitly, then press the target key. Read messages about invalid periods, interest, missing finite solutions, or indeterminate results as reasons to check the model and signs, not simply to increase display digits.

Financial calculations use real components, including when complex mode is active. A participating nonzero imaginary component produces a real-only warning; the imaginary component is not a financial input. Financial storage and recall copy complete pairs with CMPLX on and only real components with CMPLX off. A real-mode recall leaves the stored financial value intact, but puts no hidden imaginary component in X. These examples explicitly select real mode and enter their own inputs; see section 6.5.

Arriving from an HP financial calculator

SpicyRPN's RPN mode follows the HP-style TVM store/solve sequence described here, uses all four other financial quantities, lifts TVM results and financial recalls while preserving Last X and Last Y, and applies the special 0.005 cutoff to a solved n. Its five-register default, first-digit stack lift, and ENTER behavior are the SpicyRPN rules in chapter 2.

It is not a complete HP emulator. BGN is a single toggle, the monthly shortcuts are on orange n and i, and financial clear is on orange FV. Entered fractional n uses the continuous formula instead of HP's initial odd-period conventions. AMORT follows the retained-precision recurrence above; explicit RND and the actual transaction's rounding rules remain separate. ALG offers the same formulas through explicit CPT and labeled worksheets; it does not introduce a different period-rate convention. Use this chapter's complete sequences and flag checks rather than assuming that a familiar key label implies every behavior of another calculator.

Chapter 12

Cash Flows, NPV, and IRR

The cash-flow keys describe a sequence of amounts that need not be equal. Enter the initial amount, then each later receipt or payment in time order. Net present value (NPV) discounts that sequence at a rate you supply. Internal rate of return (IRR) searches for a rate at which its net present value is zero.

These commands are available in both modes on the Financial panel. Both use a cash-flow bank separate from the five financial registers, but share n and i with TVM calculations. Understanding that sharing, and the difference between a repetition count and a cash-flow index, is essential when returning to saved work.

The timing, signs, selection by n, and interpretation of NPV and IRR apply in both modes. RPN readers can follow the key sequences below; ALG readers should start with section 12.9 for worksheet entry and direct CPT calculations, then use the relevant problem sections for the shared concepts and numerical examples.

In the RPN instructions, X is the current stack position labeled S0 in the full-stack view, and Y is S1. Last X and Last Y are separate saved operands. ALG uses named worksheet fields rather than these stack positions.

12.1 Amounts at equally spaced times

CF_0 is the amount at time zero. Each later stored entry is one period farther into the future: CF_1 at time 1, CF_2 at time 2, and so on. Periods must have equal length. They can represent months, quarters, years, or another consistent interval; the calculator does not attach calendar dates to the entries.

Choose one viewpoint and keep its signs consistent. Money paid out is negative and money received is positive. For example, an initial purchase of 1000 followed by receipts of 200, 300, 400, and 500 is entered as $-1000, 200, 300, 400, 500$. If a later period includes both a receipt and an expense, enter their signed net amount for that period. Use the same monetary unit throughout.

Enter a zero for a period with no cash flow. Omitting it moves every later flow one period earlier and changes the calculation. An initial zero is also allowed when the first nonzero flow occurs after time zero. A final sale or residual amount belongs in that period's cash flow, combined with any other amount at the same time; FV is not read by NPV or IRR.

The discount rate and IRR are percentages *per cash-flow period*. Use 1 for a rate of 1 percent per month when successive entries are monthly, or 8 for 8 percent

per year when entries are annual. The calculator does not infer a period from the amount, index, or date setting. A nominal annual rate divided by 12 and an effective annual rate converted to a monthly rate are different conventions; choose the rate appropriate to the stated calculation.

Cash-flow times do not use BEGIN or END

The **BGN** setting affects TVM payments, not this explicit sequence. Cash-flow index 0 always means time zero, and index 1 means one period later. Changing BEGIN/END, MDY/DMY, or the angle unit does not shift these flows or change NPV and IRR. There is no dated-cash-flow or irregular-interval entry tool in this version.

12.2 Controls and a known starting state

Select RPN above the display. In portrait, swipe left across the top two keyboard rows to show Financial; in landscape, its keys are already visible. The second Financial row contains the five main cash-flow keys. Their functions do not use the TVM store/solve choice: CF_0 always starts a sequence, for example. In ALG, these keys open worksheets instead of taking stack operands; see section 12.9.

Table 12.1: Cash-flow controls on the Financial panel.

Keys	Purpose
CF_0	Replace the entire list with the initial amount in X; store zero in n.
CF_j	Append X as the next period's amount; set n to its index.
N_j	Use the positive integer in X as a repetition count for the last stored amount.
NPV	Calculate with flows through n, using the rate stored in i.
IRR	Search for a zero-NPV rate, using i as a starting guess if it is stored.
f **IRR**	Open IRR with bounds, the orange **IRR[a,b]** function.
g **IRR**	Open Cash flows, the blue **SHOW** function, for inspection and recall.
g **FV**	Clear the list with CL_{cf} and store zero in n.

Use real mode, a fixed stack of 5, 600 working digits, and FIX 4 for the examples. Set these as described in section 3.1 and chapter 5. Close any tool or inspector and ensure there is no pending shift or command before starting each independent sequence. f **CLEAR**, the orange function of $1/x$, clears the working stack. It leaves both the financial registers and the cash-flow list intact.

Each independent example below explicitly replaces the cash-flow list with CF_0 and sets the financial inputs it needs. The examples therefore do not depend on empty memories, but they do replace any earlier cash-flow problem. Other financial inputs such as PV, PMT, and FV can remain stored without participating in NPV or IRR.

Use explicit **STO** **i** for a discount rate or guess, and **STO** **n** for a chosen final index. This states the intention to store regardless of whether a bare TVM key would store or solve. Chapter 11 explains that separate TVM behavior.

12.3 Entering individual and repeated flows

The capacity is **1000 expanded entries, including CF0**: indices 0 through 999. Therefore a sequence can contain at most 999 future periods. Repetitions consume one entry per occurrence, just as individually entered flows do. A proposed CFj or Nj operation that would exceed capacity is rejected without appending a partial group or changing n. A valid frequency can still exceed the remaining capacity; check both the repetition count and the existing list length.

All inputs and results must be finite. NPV and IRR use real components of stored values; complex mode does not provide complex discount rates or financial roots. A discarded nonzero imaginary component produces the real-only warning when complex mode is active. Calculation retains the original stored flows rather than erasing their imaginary parts. For ordinary financial work, use real mode and deliberately enter the intended real amounts.

Type the initial signed amount and press CF_0. That key accepts an unfinished number, stores it as the sole entry, and resets n to zero. Type each later amount and press CF_j. No **ENTER** is needed between an amount and its cash-flow key, or before the next freshly typed amount: the preceding key has already finished number entry. CFj and Nj require an existing initial flow. If the bank is empty, begin with CF0; entering a subsequent flow first reports an error.

These three entry commands leave X in place after accepting it and preserve both Last registers. Ordinary digit entry still lifts the stack when the next number begins. The working stack is not the cash-flow record; amounts remain in the separate bank even after they have moved out of a fixed stack or the stack is cleared. Cash-flow writes and recalls follow the shared memory rule: with CMPLX on they copy both components, and with it off they write only the real component. This includes new repetitions made by Nj. Existing flows keep their stored components until replaced; see section 6.5.

Enter four unequal annual receipts

Use the chapter's prepared RPN settings, real mode, FIX 4, and no pending command. Enter an initial payment of 1000 and four annual receipts:

f CLEAR 1000 +/− CF_0
200 CF_j 300 CF_j
400 CF_j 500 CF_j

The final display is `500.0000`. The bank contains five amounts, at indices 0 through 4, and `n` contains 4. Press RCL n to check it: the display becomes `4.0000`. That recall lifts a value onto the stack; it does not change the cash-flow selection.

Repeat an amount for consecutive periods

After storing an amount with CF_j, type its total repetition count and press N_j. A count of 3 means three consecutive occurrences *including the one already entered*. It appends two more copies. A count of 1 adds nothing. The count must be an integer from 1 through 1000, and the expanded list must still fit the 1000-entry capacity stated above.

SpicyRPN stores the expanded period-by-period sequence, not groups with separately editable frequencies. Each copy has its own index. Pressing N_j again appends another count-minus-one copies of the last stored amount; it does not revise the size of an earlier group. Neither recalling a flow nor changing `n` changes which stored amount is the last one.

Three equal receipts followed by a different one

Use the same prepared RPN settings, with no pending prefix. Enter:

f CLEAR 1000 +/− CF_0
250 CF_j 3 N_j 400 CF_j

The resulting bank is:

Index	0	1	2	3	4
Amount	−1000	250	250	250	400

There are five stored entries covering four future periods. The final display is `400.0000`; RCL n shows `4.0000`. At an 8 percent rate, continue with `8` STO i NPV to obtain `-61.7138`.

N_j repeats the actual final stored entry even if that entry is CF_0. Using it immediately after CF0 therefore copies the initial amount into subsequent periods; it is not a way to repeat a time-zero amount at time zero. Normally, enter the first future amount with CFj before applying a frequency.

12.4 Reviewing and replacing stored amounts

In RPN, press **g** **IRR** to open the Cash flows panel. Its stepper reads CF k of m, where k is the selected index and m is the last stored index. Use its decrease and increase controls to move through the entries. The panel shows one amount at a time, with up to 32 significant digits, independently of the main display's FIX count. It is a review panel, not an editable table of amounts or frequencies.

The selector retains the previously viewed index when possible; after the list becomes shorter it is clamped to a valid index. It does not necessarily reopen at zero. The selector covers the whole stored list, even if n currently selects a shorter prefix for calculation. Moving the selector does not change n, the stack, or an unfinished number. **Close** returns without recalling anything.

Recall into X commits any active number, replaces X (display S0) with the displayed flow, and closes the panel. It does not lift or drop the stack, and it preserves both Last registers. The stored flow is unchanged. With an empty bank, the panel says No cash flows entered.; trying to recall reports an error and leaves the panel open.

Direct indexed recall

Indices begin at 0 and must identify an existing entry. If the bank contains five flows, valid indices are 0 through 4, not 1 through 5. An out-of-range, negative, or fractional index is rejected. Indexed recall is not limited to the flows currently selected by n.

For a known index, type it into X, then press **RCL** CF_j. Use the unshifted CFj key as the target. It replaces the index with that flow, saves the index in Last X, and leaves Last Y and the deeper stack unchanged. This differs from the SHOW panel's recall, which preserves both Last registers. Neither method changes n.

Replace or adjust one existing flow

To replace a flow, enter the new amount, press **ENTER**, then enter its index and press **STO** CF_j. The value comes from Y (S1); the index comes from X (S0). Storage consumes the index, leaving the new amount in X. It saves the index in Last X and preserves Last Y. The number of flows and the selected n are unchanged. Index 0 can be replaced this way without resetting the rest of the sequence.

Arithmetic storage also works. Put the adjustment in Y and the existing index in X, then press **STO**, the arithmetic operator, and unshifted CFj. For example, **STO** **−** CF_j subtracts the adjustment from the stored flow; division divides the stored flow by the adjustment. A zero divisor is rejected. Afterward the adjustment, not the new stored result, remains in X; recall the index to inspect the changed amount.

Edit just one member of a repeated sequence

Use the chapter's prepared RPN settings, with no pending command. Re-create the five-entry list so its previous contents do not matter:

f **CLEAR** 1000 **+/−** CF_0
250 CF_j 3 N_j 400 CF_j

Replace only the flow at time 2, then increase it by 25:

275 **ENTER** 2 **STO** CF_j
25 **ENTER** 2 **STO** **+** CF_j

The first store leaves `275.0000` in X; the second leaves `25.0000`. Check the stored result with 2 **RCL** CF_j: it is `300.0000`. The list is now $-1000, 250, 300, 250, 400$, with $n = 4$; only one copy changed. Continue with 8 **STO** **i** **NPV** to obtain `-18.8469`.

There is no individual insertion, deletion, or group-frequency editor. Replacing a flow with zero keeps its period and index. To remove a mistaken extra period and shift later amounts earlier, start a new sequence with CF0 and re-enter it correctly. Indexed STO cannot append an entry or create a gap; use CFj and Nj for appending.

12.5 Selecting the prefix with n

For both NPV and IRR, **n** is the index of the *last included* flow. The calculation uses the inclusive prefix $\mathrm{CF}_0, \ldots, \mathrm{CF}_n$. Thus $n = 4$ means five amounts: the initial flow and four future periods. It is not the number of groups entered with CFj, and it is not the most recently typed Nj frequency.

The selector must be an explicitly stored integer from 0 through the last existing index. A count beyond the list, a negative value, or a fraction is an error. NPV and IRR do not use a fractional TVM period count or round it into a cash-flow index.

CF0, CFj, and Nj automatically set **n** to the final stored index after their operation. To temporarily use a shorter prefix, enter its last index and press **STO** **n**. All later flows remain stored and visible in SHOW. To restore the full sequence, store

its final index again. NPV allows $n = 0$, returning CF0 alone at any valid supplied rate; IRR needs at least two selected flows.

TVM and cash flows share n and i

A TVM calculation, amortization, or another financial task can change n or i while leaving the cash-flow list intact. Conversely, adding cash flows changes the TVM period register. Before resuming either task, explicitly set its required inputs. After selecting a shorter prefix, CFj still appends to the actual end of the stored list, and Nj still repeats its actual last amount. Either operation selects the new final index; it does not overwrite or truncate the unselected tail.

12.6 Net present value

NPV expresses the selected sequence as one signed amount at time zero. With periodic percentage i, its formula is

$$\mathrm{NPV}(i) = \sum_{t=0}^{n} \frac{\mathrm{CF}_t}{(1 + i/100)^t}. \tag{12.1}$$

CF0 is already at time zero and is not discounted. A positive rate reduces the present contribution of later positive receipts. At zero interest, NPV is simply the signed sum of the selected flows. Negative rates are accepted when they are greater than −100 percent; the rate −100 and all smaller rates are rejected.

Store the percentage in i, then press **NPV**. Unlike the TVM convention for unwritten inputs, NPV requires i to be explicitly stored. If you want zero interest, store zero. It also requires a valid stored n, normally supplied by cash-flow entry. An empty list cannot be evaluated.

The result lifts into X (display S0), moving the previous X to Y (S1). It preserves both Last registers, the cash-flow list, and all five financial registers. The result is not automatically stored in PV. Repeating NPV recalculates and lifts another result. Display rounding does not change the amount used in later arithmetic.

Discount a four-year sequence at 8 percent

Use the chapter's prepared RPN settings, real mode, 600 working digits, FIX 4, and no pending prefix. Enter:

f CLEAR 1000 +/− CF_0
200 CF_j 300 CF_j 400 CF_j
500 CF_j 8 STO i NPV

The display is `127.4347`, in the same monetary unit as the inputs. The independent calculation is

$$-1000 + \frac{200}{1.08} + \frac{300}{1.08^2} + \frac{400}{1.08^3} + \frac{500}{1.08^4} \approx 127.4346540820.$$

n remains 4 and i remains 8. The positive amount is the difference between the discounted future receipts and the initial payment under these particular assumptions.

To see the selection effect using that just-entered list and the same settings, press 2 STO n NPV. The display becomes `-557.6132`: only CF0, 200, and 300 participated. Continue with 0 STO n NPV to obtain `-1,000.0000`, then restore the complete selection with 4 STO n. The omitted flows were retained throughout.

You can compare discount rates without re-entering the flows. With that full list and $n = 4$, 0 STO i NPV gives `400.0000`. Replacing the rate with 10 +/− STO i NPV gives `903.3684` at −10 percent. Explicitly restore the rate you intend to use after such comparisons.

12.7 Internal rate of return

An IRR is a periodic percentage i satisfying $\mathrm{NPV}(i) = 0$. It is a rate, not an amount and not a total percentage increase over the whole sequence. The search uses the same prefix through n as NPV and considers finite rates greater than −100 percent.

IRR requires at least two selected entries and at least one positive and one negative amount. All-zero data does not determine a unique rate; a sequence of one sign has no finite zero-NPV rate in the allowed domain. Opposite signs are necessary, but do not guarantee a solution. For example, $-100, 100, -100$ has no real rate above −100 percent that makes its NPV zero.

A stored i supplies the initial rate guess; if it is unset, the starting guess is zero. A supplied guess must itself be finite and above −100 percent. After checking the

guess, press **IRR** to search automatically. The guess is a search starting point, not a constraint that the answer equal or remain on one side of that rate. To control a specific interval, use the bounds panel described next.

IRR lifts its result into X (display S0) and preserves both Last registers. It does *not* overwrite i, nor does it change n or the stored flows. If you next press NPV, NPV therefore still uses the old rate. To check the returned rate, press **STO** **i** **NPV** immediately after IRR. This stores the retained answer, including its undisplayed digits.

Find the rate for the four annual receipts

Use the chapter's prepared RPN settings and no pending prefix. Re-enter the entire problem and choose an 8 percent starting guess:

f **CLEAR** 1000 **+/−** CF_0
200 CF_j 300 CF_j 400 CF_j
500 CF_j 8 **STO** **i** **IRR**

The result is `12.8257`, or about 12.8257 percent per year for these annual flows. Its leading digits are 12.8257269001674. i is still 8.
Continue directly with **STO** **i** **NPV**. The residual is very close to zero. It can be a tiny nonzero number shown in scientific notation even under FIX 4; the solver and the discounted sum use finite precision. Do not retype 12.8257 for this check, because that would test a rounded rate instead.

For monthly flows, a result of 1 means 1 percent per month. Under monthly compounding the corresponding effective annual percentage is $100[(1+1/100)^{12}-1]$, about 12.6825 percent, whereas 12 times the monthly rate is a nominal annual percentage. Those interpretations require a chosen compounding convention; IRR itself returns only the rate per entered period.

12.8 Bounds, multiple roots, and unsuccessful searches

With an initial outflow and only nonnegative later receipts, including at least one positive receipt, NPV decreases as the discount rate rises. With later outflows, that relationship need not hold, and more than one rate can make NPV zero. Automatic IRR returns one root found from its guess, not a list of every possible rate or a guarantee that the chosen root is unique.

The interval must bracket a sign change in NPV, unless an endpoint already evaluates to zero. A range that merely contains a root is not sufficient if its endpoint NPVs have the same sign. Use an interval around the crossing you want; a broad interval

containing several crossings does not identify all of them. An explicit bracket supplies the search interval directly instead of using i as a guess.

Press f IRR to open IRR with bounds. The lower bound must be strictly less than the upper bound, and both must be finite and greater than −100 percent. Replace the Lower % and Upper % fields with periodic percentages. The fields are typed panel inputs, not operands taken from the stack. Their previous contents can remain from an earlier use, so check both fields every time. Their first-use defaults are 0 and 2 percent. After checking both bounds, press Apply. ALG uses the separate worksheet in section 12.9, with initial bounds of 0 and 100 percent, ENTER to accept edited fields, and COMPUTE to calculate.

Successful Apply closes the panel and lifts the rate into X with the same register effects as ordinary IRR. An invalid field or unsuccessful search leaves the panel open with a message for correction. Opening the panel and closing without applying preserve the working entry. An attempted calculation can commit that entry before reporting an error, so check **ENTRY** before trying a digit correction.

Select either of two possible rates

Use the chapter's prepared RPN settings and no pending command. Enter an initial outflow, an intervening receipt, and a final outflow:

f CLEAR 100 +/− CF_0

230 CF_j 132 +/− CF_j

Press f IRR. Set Lower % to 5 and Upper % to 15, then press Apply. The result is `10.0000`. Open the same panel again, set the bounds to 15 and 25, and apply. The result is `20.0000`.
Both satisfy the same equation. With $q = 1 + i/100$,

$$-100 + \frac{230}{q} - \frac{132}{q^2} = 0 \quad \Longleftrightarrow \quad (q - 1.1)(q - 1.2) = 0.$$

The bounds distinguish the two crossings; they do not change the cash-flow data or select a different set of periods.

For the just-entered list, automatic IRR also illustrates the guess's effect. With the same settings and no pending prefix, 9 STO i IRR returns `10.0000`; 21 STO i IRR returns `20.0000`. These observed choices are not a general nearest-root rule. Bounds make the intended crossing explicit.

A failure message is not a list of roots

A root can also touch zero without changing sign. For $-100, 220, -121$, the mathematical rate 10 percent is such a double root. Bounds of 5 and 15 percent have

negative NPV at both ends and are rejected by this sign-changing-bracket method. Likewise, the two-root example above has equal-sign endpoints outside both roots; an interval enclosing both may be rejected despite containing two solutions. Trying different guesses or bounds is not a complete enumeration of roots.

A Financial solver message can report failure to bracket a sign change or failure to converge within the iteration limit. Read the message, check the selected flows and signs, then inspect NPV at candidate rates to find a meaningful sign-changing interval. A failed numerical search by itself does not prove that the mathematical problem has no root. Failed NPV/IRR calculations do not replace stored amounts or produce a new stack result, although a pending entry may have been accepted before the failure was found.

12.9 Cash-flow worksheets in ALG

Main CF0, CFj, Nj, NPV, and IRR open their corresponding worksheets. Accept each edited field with ENTER, then select the named action. Nothing executes merely because a worksheet opens or resumes.

Table 12.2: ALG cash-flow entry and calculation.

Key or operation	Fields	Action and effect
CF0	Initial cash flow	STORE replaces the entire sequence and sets n to 0.
CFj	Next cash flow	APPEND requires CF0 and adds the next amount; n becomes its index.
Nj	Frequency	REPEAT counts the original occurrence and adds copies of the latest flow; n becomes the final index.
STO CFj	Cash-flow index, Value	STORE replaces an existing entry. Arithmetic STO selects ADD, SUBTRACT, MULTIPLY, or DIVIDE. n is unchanged.
RCL CFj or blue IRR (SHOW)	Cash-flow index	RECALL supplies a labeled output; Use value inserts it in the main expression.
NPV	Last included index (n), Interest per period % (i)	COMPUTE discounts CF0 through CFn.
IRR	Last included index (n), Initial guess % (i)	COMPUTE returns a periodic percentage without overwriting i.
Orange IRR	n, initial guess, lower and upper rate %	COMPUTE searches the chosen sign-changing interval.

The operation selector provides the related financial worksheets, including indexed editing. SHOW opens indexed recall without reading or inserting a value automatically. On its first visit the index is zero; subsequent visits retain the worksheet's accepted index or draft. Enter the desired index in the worksheet, press ENTER, then RECALL. Zero is CF0's index; an index must identify an existing entry. Repetition is expanded storage, subject to the same 1000-entry limit as RPN.

The ALG bounded-IRR worksheet initially offers 0 and 100 percent. Its accepted bounds and edits are retained, so review both every time. These defaults differ from the RPN bounds panel's initial 0 and 2 percent. Both modes require a valid sign-changing interval as described in section 12.8.

Before a direct CPT calculation, finish or clear a pending compound expression and set the intended discount rate or initial guess. Mapped financial fields read the current register values, including zero for an unset register. Thus ALG NPV can use an unset i as zero; RPN NPV requires an explicitly stored i. CPT followed by NPV or IRR performs a direct calculation from the shared cash-flow bank and current n and i. Its successful result becomes the displayed completed answer and ANS. The unprefixed NPV and IRR keys instead open worksheets, allowing labeled input review and retention of the main expression. IRR never replaces i merely because it was computed.

Financial fields accepted with ENTER update their mapped registers. Other accepted fields remain worksheet inputs until the named action validates the complete operation. A failure cannot partly replace a cash-flow list. On success the declared primary output is the stored or recalled amount, frequency, NPV, or IRR. CF0, CFj, and Nj additionally show the updated last index; indexed replacement, indexed recall, NPV, and IRR return only their primary result. Browsing outputs does not alter ANS. Canceling a draft is not undo for completed storage or repetition.

For the sequence $-1000, 200, 300, 400, 500$, STORE -1000 in CF0, then APPEND each of the four later values. The last index is 4. Open NPV, accept n = 4 and i = 10, and COMPUTE. The result is approximately 71.78471416. This is the same periodic calculation described earlier, without using any RPN stack register.

12.10 Comparing amount and rate results

NPV and IRR answer different numerical questions. NPV is a monetary amount at a supplied rate; IRR is a rate derived from the sequence. Multiplying every flow by the same positive factor multiplies NPV by that factor but leaves IRR unchanged. Comparisons should therefore retain each sequence's scale, timing, and the rate used for discounting.

Two scales of investment over two periods

Use the chapter's prepared RPN settings, real mode, FIX 4, 600 working digits, and no pending command. Project A pays 100 now and receives 144 at the end of period 2, with nothing at period 1. Enter:

f CLEAR 100 +/− CF_0 0 CF_j
144 CF_j 10 STO i NPV

The NPV is `19.0083`. Press IRR to obtain `20.0000` percent per period. Record those results; the next CF0 replaces this list.
Project B pays 1000 now and receives 1322.5 at the same time 2:

f CLEAR 1000 +/− CF_0 0 CF_j
1322.5 CF_j 10 STO i NPV

The NPV is `92.9752`. Press IRR to obtain `15.0000` percent per period. The independent checks are $-100 + 144/1.1^2$ and $-1000 + 1322.5/1.1^2$ for NPV; $144/100 = 1.2^2$ and $1322.5/1000 = 1.15^2$ give the rates.

Project A has the larger percentage rate in this example, while B has the larger monetary NPV at 10 percent and requires a larger initial amount. The outputs describe those distinct properties; they do not supply any missing assumptions about costs, risk, available capital, or the accuracy of the entered receipts. The explicit zero at period 1 is part of each calculation, not an unused memory slot.

12.11 Clearing and numerical limits

Choose a clear according to the data you intend to discard. Table 12.3 distinguishes the cash-flow list from the financial registers and the working stack. All sequences assume there is no existing pending command or shift.

CF0 lights **CF** because it stores an entry, and **FIN** because it stores $n = 0$. CLcf removes **CF**, but leaves **FIN** present because that zero in n is still stored. A zero cash flow also remains occupied and keeps CF lit. The financial MCLR group removes both banks and both flags. For unfinished-entry handling, retained settings, and prefix cancellation, see section 6.10.

NPV uses the current working precision, not the displayed number of places. IRR is an iterative result with a separate convergence rule. The current solver searches in $\ln(1 + i/100)$, with a standard interval tolerance of 10^{-30} scaled to the size of that transformed variable and a limit of 200 iterations per search stage. The bounds panel has no tolerance setting. A working precision of 600 digits consequently does not promise 600 correct IRR digits; rounding, sensitivity, the stopping rule, and

Table 12.3: Clearing and replacing cash-flow work in RPN.

Sequence	Effect on stored financial work
Amount, CF_0	Replace all flows with that initial amount and store $n = 0$. Other financial registers remain.
g FV: CLcf	Remove all flows and store $n = 0$. Preserve i, PV, PMT, and FV.
f FV: CLfin	Remove all five financial registers; retain the cash-flow list. Restore n before using the list, and i before NPV.
g STO n	MCLR's financial group: remove all five financial registers and all cash flows.
f CLEAR	Clear the RPN stack and unfinished entry; preserve both financial banks.

iteration limits still matter. See section C.9 for the full stopping rule and chapter 5 for the precision of retained inputs.

When a result is unexpected, first inspect the full stored sequence, then recall n and i. Check for an omitted zero period, an amount with the wrong sign, a repeated count that included extra copies, a shorter selected prefix, or a rate left by a different financial task. For IRR, also check the guess or bounds and whether the cash-flow signs permit multiple roots. The bank survives stack clearing, mode changes, and relaunch; those actions do not prepare a new cash-flow problem.

Chapter 13

Additional Financial and Date Calculations

Simple interest, dates, bonds, and depreciation share the Financial panel with TVM and cash flows. Their inputs have different meanings, however: the financial n register can hold days or an asset's useful life, and i can hold an annual interest rate or a declining-balance factor. Set the required inputs for the particular operation rather than reusing a previous problem's values by accident.

These operations are available in both modes. The input meanings, date encoding, financial conventions, and formulas below apply to both. RPN readers can follow the stack procedures and worked key sequences. ALG readers should start with section 13.12 for the labeled worksheets, then read the relevant simple-interest, date, bond, or depreciation section for its assumptions and interpretation. ALG preserves the main expression while a worksheet is open.

In the RPN procedures, X is the current stack position labeled S0 in the full-stack view, and Y is S1. Last X and Last Y are separate saved operands. Financial registers and ALG worksheet fields are separate inputs, not names for these stack positions.

13.1 Controls and preparation

In portrait, swipe left across the top two keyboard rows to show Financial. In landscape, both panels are visible. The operations in this chapter occupy the alternate legends listed in table 13.1. Press the appropriate shift, then the physical main key in the first column. The main function does not run when you select its alternate legend.

The blue INT on i means *simple interest*. It is different from orange INT on ENTER in RPN or $1/x$ in ALG, which truncates a number to its integer part. Identify the physical key as well as the legend when selecting either function.

Use RPN, real mode, a fixed stack of 5, and 600 working digits for the stack examples. Select FIX 4 for money and depreciation, and FIX 6 when reading dates. The preparation procedures in section 3.1 and chapter 5 set these conditions. In RPN, 4 f 7 selects FIX 4 and 6 f 7 selects FIX 6. Each example starts with no pending shift or command and includes a stack clear, f CLEAR, on the $1/x$ key. Angle units and BEGIN/END payment timing do not affect these tools.

Table 13.1: Financial and date controls used in this chapter.

Main key	Orange function	Blue function
i	12÷, covered in chapter 11	INT: simple interest
PV	PRC: bond price	YTM: bond yield
CF_0	DATE: add days	ΔDAYS: two day counts
CF_j	MDY: month/day/year	DMY: day/month/year
N_j	SL: straight line	SYD: sum of years' digits
NPV	DB: declining balance	DB→SL: declining balance with crossover
PMT	AMORT, covered in chapter 11	RND: round the current value

The examples explicitly store every financial input they use. For example, 90 STO n stores 90 regardless of what the next bare n would otherwise do. An unset required register is an error for these additional RPN financial tools, even when its ordinary recall would show zero. Store a required zero explicitly. Unused financial registers do not participate. This differs from a TVM solve's treatment of unset inputs in chapter 11.

Clearing the stack leaves financial registers and cash flows intact; the explicit stores in each example replace its required inputs. For bank clearing, use section 6.10. Before returning to TVM or cash flows, review the register changes summarized in section 13.13.

13.2 Simple interest on 360- and 365-day bases

Simple interest is proportional to principal, annual rate, and elapsed days; it does not compound interest into principal. The routine requires finite inputs and nonnegative days; it does not apply TVM's greater-than-minus-100-percent rate restriction. A positive PV reverses the signs of the returned principal and interest. A negative annual rate is also accepted and reverses the interest sign. A zero rate or zero days returns zero interest while retaining the principal result.

Set the three registers in table 13.2, then press g i, selecting its blue INT function. The command reads these registers, not the visible stack number.

Writing $P = -PV$, $d = n$, and $j = i/100$, the two interest amounts are

$$I_{360} = Pj\frac{d}{360}, \qquad I_{365} = Pj\frac{d}{365}.$$

Both are returned by one command. The main result in X (S0) is I_{360}; Y (S1) holds P; S2 holds I_{365}. Thus + immediately afterward adds the 360-day interest to the returned principal. The financial registers themselves are unchanged.

Table 13.2: Required simple-interest inputs. PMT and FV are unused.

Register	Meaning
n	Number of days, nonnegative and finite. A fractional number of days is accepted.
i	Annual percentage rate; enter 6 for 6 percent.
PV	Signed principal. Enter a negative amount for money initially paid out when the returned principal and interest should be positive.

The traditional name "exact interest" for the 365-day result means that fixed denominator. It does not select a leap-year denominator of 366, an actual/actual year fraction, or a special business-day calendar. Similarly, the 360-day denominator does not turn an actual elapsed-day count into a 30/360 count. Choose the days to store separately; the date-difference command below supplies both kinds of day count.

This three-result operation discards the old stack X and inserts the three results above the old Y. It saves the old stack X and Y in the Last registers; those saved values are not copies of the financial inputs. An unlimited stack must have at least two operands, although their values are not used by the formula. A fixed depth of at least 3 is needed to retain all three results; the chapter's depth of 5 is sufficient.

Interest on a 90-day advance

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. Show the Financial panel. Replace the three required inputs, then calculate:

f CLEAR 90 STO n 6 STO i
1000 +/− STO PV f CLEAR g i

The results are X = `15.0000`, Y = `1,000.0000`, and S2 = `14.7945`. The checks are $1000(0.06)(90/360) = 15$ and $1000(0.06)(90/365) = 14.7945205479\ldots$. Inspect S2 with the full-stack view if desired, then close it. Press + to obtain `1,015.0000`, the principal plus 360-day interest.

13.3 Entering and reading calendar dates

Select date order before entering or calculating with dates. Switching MDY/DMY changes how numbers will be interpreted; it does not rewrite a number already on the stack, in a memory, or still being typed. A number such as `2.032026` changes its interpretation from February 3 to March 2 when the order changes. A value valid in one order can be invalid in the other.

Dates are entered as numbers with a packed decimal layout. The first component is before the decimal point; the second occupies two places after it, followed immediately by the four-digit year. There are no slashes or separate year fields.

Table 13.3: Date order and entry of 3 February 2026.

Flag	Select with	Layout	Example
MDY	f CF_j	Month.DDYYYY	`2.032026`
DMY	g CF_j	Day.MMYYYY	`3.022026`

Include the leading zero in a one-digit second component: February 3 is `2.032026` in MDY, not `2.32026`. Use all four year positions, including leading zeros for early years. For example, January 1 of year 1 is `1.010001` in MDY. Leading zeros before the decimal point are unnecessary. Extra trailing zeros leave the numerical value unchanged, but extra nonzero fractional digits do not define a valid date.

The calendar is Gregorian throughout years 1 through 9999. A year divisible by 4 is a leap year, except a century must also be divisible by 400. Thus 2000 has February 29 and 1900 does not. Impossible dates such as February 30, April 31, or February 29 in a non-leap year are rejected rather than rolled into the following month. Year zero, negative dates, and years beyond 9999 are outside the supported range.

Show all six fractional date digits

A date result is an ordinary stored number, so FIX 4 can hide part of its year in the calculator display. Use FIX 6 when reading an encoded date there. The ALG Result date worksheet output automatically uses six fractional places; Use value inserts its full value into the main expression, whose ordinary display settings apply again. Changing the display count leaves its value intact; applying **RND** can destroy the date by rounding away required digits. Date values carry no separate calendar tag and are not reformatted automatically when MDY/DMY changes.

Dates have no time of day, time zone, or daylight-saving adjustment. These functions count civil calendar days. They do not select working days, omit weekends or holidays, or report a weekday. The Time family in **CONV** instead converts fixed durations such as hours and 24-hour days; see section 9.7.

13.4 Adding or subtracting calendar days

The day count must be an integer from −3 652 058 through 3 652 058, and the resulting date must still lie within years 1–9999. The count's broad numerical range does not permit crossing either calendar endpoint. Adding one day to December 31, 9999, or subtracting one from January 1, year 1, is rejected.

Put the starting date in Y (S1) and the number of days in X (S0), then press f CF_0, the **DATE** function. Enter the date first, press ENTER, enter the day count, and select DATE. A positive count moves forward; a negative count moves backward. Zero returns the same date.

Cross a leap day in either direction

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 6, with no pending prefix. For month/day/year, press:

f CF_j f **CLEAR** 2.282024 **ENTER** 2 f CF_0

The result is 3.012024: two days after February 28, 2024 is March 1, because February has 29 days that year.
For a separate example in day/month/year, select DMY and calculate one day before March 1:

g CF_j f **CLEAR** 1.032024 **ENTER** 1 +/− f CF_0

The result is 29.022024, or February 29, 2024. Financial-register contents are irrelevant to both calculations.

DATE consumes both operands and returns one date in X (display S0), like an ordinary binary operation. The prior day count is saved in Last X and the prior starting date in Last Y. The returned date is encoded in the currently selected order. It can be used immediately as the starting date for another offset: type the new count and select DATE. Typing that new count lifts the returned date into Y.

13.5 Actual and 30/360 day counts

There are several conventions called 30/360. SpicyRPN applies the following specific rule to a first date (Y_1, M_1, D_1) and second date (Y_2, M_2, D_2):

$$d_1 = \begin{cases} 30, & D_1 = 31 \text{ and } D_2 \geq 30, \\ D_1, & \text{otherwise,} \end{cases} \qquad d_2 = \min(D_2, 30),$$

$$D_{360} = 360(Y_2 - Y_1) + 30(M_2 - M_1) + d_2 - d_1.$$

There is no additional February-end adjustment. Do not assume this matches another system merely because its label also says 30/360. For January 31 to February 28, 2023, the rule gives 27, while the actual count is 28. In reverse it gives −28, not −27. The adjusted count is therefore not generally antisymmetric at month-end; reversing the dates and negating the answer need not be equivalent.

For a difference, put the first date in Y (S1) and the second in X (S0), then press g CF_0, the **ΔDAYS** function. It consumes the two dates and returns the actual

elapsed days in X (S0) and the calculator's 30/360 count in Y (S1). The first date is not counted as an additional day: equal dates return zero. Reversing the dates negates the actual elapsed count. Both input dates are saved in the corresponding Last registers.

Compare a calendar interval with a 30/360 interval

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 6, with no pending prefix. Select MDY and enter January 31 through March 31 of the non-leap year 2023:

f CF_j f CLEAR 1.312023 ENTER 3.312023 g CF_0

The main result is `59.000000` actual days; the Y detail line shows `60.000000` for 30/360. February contributes 28 calendar days and March contributes 31, while the adjusted calculation treats the interval as two 30-day months. No financial inputs are required.

To use the actual count for simple interest, press STO n after the difference. To use the adjusted count instead, first press $x \leftrightarrow y$, then STO n. Set the annual rate and signed principal as described in section 13.2. Keep the day-count convention and the interest denominator as two separate choices.

13.6 Bond inputs and coupon assumptions

The bond functions calculate price or yield for a fixed coupon paid semiannually, with redemption at 100 per 100 of face value. They construct the coupon dates from maturity; there are no separate fields for coupon frequency, issue date, first coupon, redemption value, or an irregular coupon schedule. BEGIN/END and the 30/360 result from ΔDAYS do not change the bond convention.

For either bond function, enter settlement first, press ENTER, then enter maturity. Settlement goes in Y (S1) and maturity in X (S0). Both follow the selected MDY/DMY order, and maturity must be later than settlement. Table 13.4 identifies the required financial registers. Store the annual coupon percentage itself in PMT, not the amount of one semiannual coupon and not a signed loan payment.

Coupon percentage must be finite and nonnegative. A zero coupon is valid, but must still be stored in PMT. Yield for PRC must be finite and greater than −200 percent, so its semiannual growth factor is positive; negative yields above that boundary are permitted. YTM requires a finite, strictly positive clean price. The annual yield is quoted as twice the semiannual percentage rate, not as an effective annual rate. Thus yield 6 means 3 percent per half-year; do not divide it by two before storing it in i.

Table 13.4: Bond inputs and updates. Financial n and FV are unused.

Operation	Required registers	Results and update
f PV: PRC	PMT = annual coupon percent; i = annual yield percent.	X (S0) = clean price per 100; Y (S1) = accrued interest per 100. Clean price is also stored in PV.
g PV: YTM	PMT = annual coupon percent; PV = positive clean price per 100.	X (S0) = annual yield percent, also stored in i.

How coupon dates and accrued interest are determined

Coupon dates step backward in six-month intervals anchored to the original maturity date. A maturity on the last day of its month produces month-end coupon dates; August 31, for example, produces February 28 or 29 and earlier August 31 dates without drifting after February. Otherwise the maturity's day is retained when possible, and shortened to the month's last day when necessary. The calculation also needs the coupon date at or before settlement, so the schedule must not extend before year 1.

Let $C = \mathrm{PMT}/2$ be the semiannual coupon per 100, E the actual days between the surrounding coupon dates, and d the actual days from settlement to the next coupon. The fraction to the next coupon is $w = d/E$; accrued interest is $A = C(1 - w)$. On a coupon date, settlement is treated as the start of the next coupon interval: $w = 1$, accrued interest is zero, and the coupon on settlement itself is not a future payment.

Clean price excludes accrued interest. Adding the returned accrued interest gives the full or *dirty price* per 100 under this model. To scale either amount to a different face value, multiply by that face value divided by 100. The calculator does not add charges, taxes, or other settlement adjustments.

13.7 Calculating bond price and yield

For two or more remaining coupons, the full price is the discounted value of the coupon payments and redemption:

$$P_{\mathrm{dirty}} = \sum_{k=0}^{N-1} \frac{C}{(1+r)^{w+k}} + \frac{100}{(1+r)^{w+N-1}}, \qquad r = \frac{i}{200}.$$

Here N is the number of future coupon payments constructed from the dates, not the financial n register. PRC returns $P_{\mathrm{clean}} = P_{\mathrm{dirty}} - A$. It replaces the two date

operands with clean price in X (S0) and accrued interest in Y (S1), and stores the clean price in PV. Both original dates become the Last X/Y operands.

Price a bond between coupon dates

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. Set a 6 percent annual coupon and yield, select MDY, then enter settlement April 1, 2024 and maturity January 1, 2026:

f CLEAR 6 STO PMT 6 STO i f CF_j

f CLEAR 4.012024 ENTER 1.012026 f PV

The main display is `99.9889`, the clean price per 100. Y contains accrued interest `1.5000`. The surrounding coupon interval is January 1 through July 1, 2024: 182 actual days, with 91 days from settlement to the next coupon. There are four future coupons, so $w = 1/2$, $C = 3$, and $A = 1.5$.
Press + to show the dirty price, `101.4889`. The financial PV still holds the unrounded clean price 99.9889156509 .... Its check is $100\sqrt{1.03} - 1.5$; the dirty price is $100\sqrt{1.03}$.

To check this price by calculating its yield, continue with the same MDY setting and retained PMT and PV. Clear the stack and re-enter `4.012024` ENTER `1.012026` g PV. YTM displays `6.0000`. PV still contains the unrounded clean price even if you added accrued interest on the stack above. Do not store that dirty price in PV for this inverse calculation.

YTM reverses this pricing relationship. It consumes the two dates, returns the annual percentage yield in X (S0), and stores it in i. The coupon register PMT and clean price PV remain unchanged. A one-period yield uses the direct formula below; longer schedules use a numerical solver. The latter has finite convergence tolerances and work limits; 600 working digits do not promise 600 correct yield digits. A failure to find a usable result reports a financial-solver error. YTM provides no user-entered guess or bracket, and does not use an old i as its guess.

Recover an annual yield from a clean price

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. For a 6 percent coupon bond priced at 100, settling on January 1, 2024 and maturing January 1, 2026, press:

f CLEAR 6 STO PMT 100 STO PV f CF_j

f CLEAR 1.012024 ENTER 1.012026 g PV

The result is `6.0000`, or 6 percent annual yield, and i now contains that result. Four coupons of 3 and redemption of 100 discounted at 3 percent per half-year have value 100 on this coupon date. Accrued interest is zero.

The last coupon period

With exactly one future coupon, SpicyRPN uses simple yield over the remaining fraction of a half-year:

$$P_{\text{clean}} = \frac{100 + C}{1 + rw} - A.$$

It does not use the fractional compound discount factor $(1+r)^{-w}$ in this case. This distinction can produce a clean price slightly different from 100 between coupon dates even when coupon and yield percentages are equal.

Price the final half-year fraction

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. Set PMT and i explicitly, select MDY, and enter April 1 to July 1, 2024:

f CLEAR 6 STO PMT 6 STO i f CF_j

f CLEAR 4.012024 ENTER 7.012024 f PV

X displays `99.9778` and Y displays accrued interest `1.5000`. Here $C = 3$, $r = 0.03$, and $w = 1/2$, so the clean price is $103/1.015 - 1.5 = 99.9778325123\ldots$.

Before comparing a result with a quoted bond figure, match its coupon frequency, date schedule, yield convention, price basis, and redemption assumption. There is no input for a call date, a call price, irregular first or last coupons, or a different day-count basis. Changing n or FV cannot supply those missing conventions: both registers are ignored by PRC and YTM.

13.8 Preparing a depreciation calculation

Depreciation allocates an asset's depreciable cost across its useful life. The tools here return one requested year's amount and the remaining depreciable value after that year. They do not record an asset ledger or change the stored cost as each year is requested.

All inputs must be finite. PMT is unused. SL and SYD do not read i; its previous contents cannot affect those two methods. The result in X (S0) is depreciation for year j alone. Y (S1) is the remaining *depreciable* value after that year, namely book value minus salvage. Add salvage to that Y result when you need the ending book value itself.

A depreciation command lifts two results above the original year: X holds the amount, Y holds the remaining depreciable value, and S2 still holds j. It preserves both Last registers and all financial inputs. To request another year, type that new year and select the method; clearing the working stack first makes the result

Table 13.5: Depreciation inputs for all four methods.

Location	Meaning
n	Useful life $L > 0$, in years. A fractional life is allowed; see section 13.11.
PV	Original cost K, nonnegative and at least the salvage value. Use a positive asset cost, not a signed TVM outlay.
FV	Salvage value $S \geq 0$. Store zero explicitly when no salvage value is required.
i	Positive declining-balance factor in percent, needed only for DB and DB→SL. Enter 200 for double declining balance, not 2 or 0.2.
X (S0)	Requested year j, an integer starting at 1.

positions easier to inspect. A fixed stack of at least 3 retains both results and the year. There is no automatic year increment.

Each request calculates from the original cost and useful life. Asking for year 4 does not require calculating years 1–3 first, and asking for year 2 twice does not depreciate the asset twice. If cost equals salvage, both results are zero. A requested integer year beyond $\lceil L \rceil$ also returns two zeros.

13.9 Straight line and sum of years' digits

For an integer useful life L, straight line spreads the base $B = K - S$ equally across all years:

$$D_j = \frac{B}{L}, \qquad R_j = B - j\frac{B}{L} \quad (1 \leq j \leq L).$$

Select SL with f N_j. The symbol R_j denotes the remaining depreciable value in Y, not the book value including salvage.

Sum of years' digits assigns the descending weights $L, L-1, \ldots, 1$, whose sum is $L(L+1)/2$. It therefore allocates more depreciation to earlier years:

$$D_j = B\frac{L - j + 1}{L(L+1)/2}.$$

Select SYD with g N_j. The remaining result subtracts all allocations through year j from the original base, even when that is the first year you request.

Compare year 2 under SL and SYD

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. Set a cost of 1000, salvage of 100, and life of 5 years:

f CLEAR 5 STO n 1000 STO PV 100 STO FV

For straight line, press f CLEAR 2 f N_j. X is `180.0000` and Y is `540.0000`: the annual amount is $900/5 = 180$, and $900 - 2(180) = 540$ remains depreciable. Ending book value is $540 + 100 = 640$.
For SYD with the same stored inputs, press f CLEAR 2 g N_j. X is `240.0000` and Y is `360.0000`. The year-2 weight is 4 out of $5+4+3+2+1 = 15$, giving $900(4/15) = 240$. Year 1 used 300; after both years, $900-300-240 = 360$ remains. S2 contains `2.0000` after either command.

13.10 Declining balance and straight-line crossover

A positive factor above the usual 100 or 200 is accepted. The salvage cap still applies, so a large factor can exhaust the depreciable base early. The DB methods recalculate the intervening annual steps and limit an active requested year to 100 000. A year beyond the useful life returns zeros without needing those steps. Raising working precision does not increase this iteration bound.

For declining balance, the annual rate is the stored percentage factor divided by the useful life:

$$q = \frac{i/100}{L}.$$

With a 5-year life and factor 200, $q = 0.4$. The ordinary DB amount is the opening book value times q, capped so that book value never falls below salvage. Use f NPV to select DB. Its factor is a depreciation factor, not an interest yield. The final year writes off the entire remaining depreciable balance, even if that differs from the usual declining-balance amount.

DB→SL, on g NPV, compares the ordinary declining amount with a straight-line allocation of the remaining depreciable value over the remaining useful life. It uses the larger amount, again capped at the remaining depreciable balance. This allows a transition to straight line before the final year. The comparison is made as the schedule is calculated; there is no separate switch-year input or stored crossover flag.

Use a 200 percent declining factor

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. Initialize all required inputs:

f CLEAR 5 STO n 1000 STO PV
100 STO FV 200 STO i f CLEAR 2 f NPV

X displays `240.0000`; Y displays `260.0000`. Year 1 depreciates $1000(0.4) = 400$, leaving book value 600. Year 2 depreciates $600(0.4) = 240$, leaving book value 360, or 260 above salvage.
With the same inputs, press f CLEAR 5 f NPV. Year 5 returns `29.6000` and `0.0000`. It writes off the balance above salvage after the earlier years; ending book value is 100, not zero.

Table 13.6: Annual amounts for cost 1000, salvage 0, life 5, factor 200. The crossover changes the last two years.

Year	DB amount	DB remaining	DB→SL amount	DB→SL remaining
1	400.0	600.0	400.0	600.0
2	240.0	360.0	240.0	360.0
3	144.0	216.0	144.0	216.0
4	86.4	129.6	108.0	108.0
5	129.6	0.0	108.0	0.0

Find the year when crossover changes the amount

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. This example changes salvage to zero:

f CLEAR 5 STO n 1000 STO PV
0 STO FV 200 STO i f CLEAR 4 g NPV

X and Y both display `108.0000`. At the beginning of year 4, book value is 216 and two years remain. Declining balance would allocate $216(0.4) = 86.4$, but straight line allocates $216/2 = 108$, so the latter is selected.
To reproduce either method's column in table 13.6, retain these initialized registers. Before each row press f CLEAR, enter the listed year, and select f NPV for DB or g NPV for DB→SL. Read the amount in X and the remaining value in Y; the table omits display trailing zeros.

13.11 Fractional useful lives and rounding

A fractional life produces a final partial year numbered $\lceil L \rceil$; the year argument itself must still be an integer. The four methods have explicit conventions for this case. There is no start-date, midyear-placement, or first-year allowance input. Supplying life 2.5 means the modeled life lasts two full years and a final half-year, rather than selecting a tax or accounting rule.

SL uses $(K - S)/L$ for each full year and multiplies that annual amount by the duration of the last partial year. SYD uses the descending weights $L, L-1, \ldots, L-m+1$, where $m = \lceil L \rceil$, and divides by their sum. For life 2.5, the weights are 2.5, 1.5, and 0.5, summing to 4.5. It does not round life up and use the integer-life weights 3, 2, and 1. Both DB methods write off all remaining value above salvage in the final, possibly partial, year. Crossover before that final year uses the actual remaining fractional life in its straight-line comparison.

Allocate the final half-year

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix. Set cost 1000, salvage 100, and life 2.5:

f CLEAR 2.5 STO n 1000 STO PV 100 STO FV

Press f CLEAR 3 f N_j for SL. The amount is `180.0000`, since $(900/2.5)(0.5) = 180$. Y is `0.0000`: salvage has been reached.
For SYD, press f CLEAR 3 g N_j. The amount is `100.0000`, since $900(0.5/4.5) = 100$, again with zero remaining in Y. Both commands retain year 3 in S2.

The tools do not round annual depreciation to the displayed decimal places. Changing FIX therefore changes the readout without changing the schedule or stored cost. Explicit RND changes only the working value in X; it does not replace PV, FV, or the internally recomputed schedule. See section 5.6 for its rounding and saved-operand rules.

13.12 Financial and date worksheets in ALG

Select the same shifted Financial-panel legend as in RPN to open its worksheet. An unset financial register initially supplies zero to its mapped worksheet field; opening a worksheet does not store its initial values. Review the intended inputs, including any required zero, and accept each with ENTER before COMPUTE. Fields naming financial registers store those registers on acceptance. Date and year arguments remain worksheet inputs. Unlike the RPN commands described earlier, ALG computation does not require that these initial register values already have

occupied storage entries. The shared controls, ANS rules, and cancellation behavior are in section 4.12.

Table 13.7: ALG additional financial worksheets. The first listed output is primary.

Legend	Accepted inputs	Outputs
INT, blue i	Days (n), Annual interest % (i), Present value (PV)	Interest on a 360-day year, Principal, Interest on a 365-day year.
DATE	Start date, Days	Result date.
ΔDAYS	Start date, End date	Actual days, 30/360 days.
PRC	Settlement date, Maturity date, Annual coupon % (PMT), Annual yield % (i)	Price per 100, Accrued interest; price is also stored in PV.
YTM	Settlement date, Maturity date, Annual coupon % (PMT), Price per 100 (PV)	Annual yield %; also stored in i.
SL, SYD	Year, Life (n), Cost (PV), Salvage (FV)	Depreciation, Book value less salvage.
DB, DB→SL	The same depreciation fields, plus DB factor % (i)	Depreciation, Book value less salvage.

Dates use the selected MDY or DMY encoding, not locale-dependent text. The date-order keys change that shared setting. Bonds retain semiannual coupons, redemption at 100, and actual coupon-period fractions; no frequency or day-count selector is added. Simple interest and bond inputs use annual percentages as labeled, while TVM and NPV continue to use periodic rates. Depreciation retains the four existing methods, including their fractional-life conventions.

COMPUTE validates the complete operation before committing output-bank changes. It updates ANS from the primary output. Select another result without changing ANS, then use Use value if that result belongs in the main expression. CANCEL returns to the suspended expression and drops unaccepted edits; financial values accepted earlier and successful computations remain stored. Relaunch retains the worksheet and its results without calculating automatically. Successful equals and ENTER update ANS with the field value as well. All these ALG operations preserve the saved RPN stack and Last registers.

13.13 Retained inputs, results, and error recovery

The shared financial bank persists across these tools, TVM, cash-flow work, mode changes, and relaunches. In particular, a bond price replaces PV, a bond yield replaces i, and storing a depreciation life or a day count replaces n. Since cash-flow

calculations also use n as their selection index, leaving the cash-flow list untouched does not preserve its previous selection. Reinitialize the relevant inputs when changing tasks.

Simple interest and depreciation leave their financial inputs unchanged. Date arithmetic needs no financial input and changes no memory bank. PRC and YTM make the one register update stated in table 13.4; other financial registers and the cash-flow list remain unchanged. A successful store lights **FIN**, including an explicitly stored zero. Occupancy flags report stored data, not which interpretation of n or i is appropriate.

All numerical operations in this chapter finish an active RPN number before calculating. Dates and bond operations save both input dates in the Last registers; DATE instead saves its day count and start date. Simple interest saves the old stack operands as described earlier. Depreciation preserves both Last registers. These recovery registers are not a backup of financial memory and cannot undo a replaced PV or i.

After an input or domain error, read the status message and inspect the stack and required registers. An unfinished number may already have been accepted, even though no result was produced. Correct an invalid financial value with explicit STO; for a date-pair error, clear and re-enter the pair in the selected order. A missing-register message requires storing the input, including a deliberate zero when appropriate. More display digits do not repair an invalid date, reversed settlement and maturity, a fractional year argument, or an unsuitable financial convention.

These tools use real components only. With complex mode enabled, discarded nonzero imaginary components produce a warning. For retained complex data or saved-state issues, see section 8.7 and chapter 14.

Part IV

Reference and Problem Solving

Chapter 14

Saved Work, Clearing, and Troubleshooting

SpicyRPN resumes the calculator you left, including unfinished work. That is useful when a calculation spans several sessions, but it also means that reopening the application is not a fresh start. An old financial input, a pending storage command, or an unfinished exponent can still determine what the next key does.

When something looks wrong, first read the mode, flags, and status message. Then identify whether you need to finish an entry, cancel a command, correct a calculation, or clear a particular bank. These are different operations. This chapter provides a way to choose among them without discarding unrelated work.

14.1 What is saved automatically

The application saves actions as you perform them, including individual entry keystrokes. There is no separate Save command and no need to press ENTER merely to preserve a partly typed number. Successful actions save their numerical and presentation changes together. The application does not depend on receiving a final quit notification to save the last key.

Saved work belongs to the application on the current device. It is held in a local database, not synchronized between your Mac, iPhone, and iPad by a calculator account. The current interface has no named calculation files, export/import command, cloud synchronization, or user-facing history browser. Keep the original observations, transaction inputs, and any results you need outside the calculator as appropriate to your work.

Table 14.1: State retained across closing and reopening the application.

Area	Retained work
RPN workspace	Stack values and order, fixed or unlimited depth, Last X and Last Y, and unfinished numeric entry.
ALG workspace	Expression, unfinished number, current value, completed-result state, and the last successful ANS.
Complex values	Both components of existing stored values, even with complex mode off; unfinished component drafts and the selected entry component. New real-mode assignments have zero imaginary part.

Area	Retained work
Shared settings	Working precision, notation and display digits, angle unit, date order, complex mode, rectangular/polar choice, and BEGIN/END payment timing.
Memory banks	Ordinary registers, associative pairs, statistical accumulators, financial registers, and the expanded cash-flow list.
Pending input	Orange or blue shift, STO and its arithmetic modifier, RCL, HYP/INVHYP, ALG FUNC, CPT and FIX prompts, and MCLR selection.
ALG worksheets	Active worksheet, accepted inputs, exact field drafts, selected field, labeled results, and selected output. Numeric values keep their components and precision, without their full operation history. Relaunch does not evaluate a draft or execute its operation.
RPN financial readiness	Whether the next bare TVM key is ready to store or solve, as explained in section 11.4.
Presentation	Selected keyboard panel, digit-ruler preference, main-number and complex-component scroll offsets, open tool and its input fields, and saved calculation or tool errors.

The full-stack inspector is temporary: it closes when the application leaves the active state, and does not reopen merely because it was open before. Open it again with the display triangle to inspect the retained stack. Window dimensions can change the layout and scientific-notation fallback without changing a value. A saved horizontal scroll position is a viewing preference, not a stored numerical change.

The database also retains information about how reachable values were calculated. That information does not provide an undo button, an earlier version of each memory bank, or a browsable calculation tape. In particular, Last X and Last Y hold saved previous RPN operands, distinct from the current stack positions X (S0) and Y (S1). LAST X/Y recall those operands under their specific rules; they do not restore a complete earlier calculator state.

14.2 Independent workspaces and shared data

RPN and ALG have independent working areas. Selecting the other mode does not evaluate an expression, finish a number, copy the displayed value, or clear the workspace you leave. Return to the earlier mode to continue its entry. RPN's Last registers and ALG's ANS are independent recovery aids, not two names for one previous result.

The settings and memory banks in table 14.1 are shared. Storing a new value in ordinary R2 in ALG changes what RPN later recalls from R2. Changing working precision or the display count in either mode affects subsequent work in both. Changing DEG to RAD changes the shared angle choice, but does not rewrite a trigonometric function already inserted into an ALG expression; that function retains its original unit. See section 4.5 and chapter 5.

An ordinary mode selection cancels active shifts and pending command prefixes. For RPN, it also makes the next bare TVM key a store. Consequently, switching away and back is not equivalent to simply pausing immediately after a financial store. When resuming a financial task, use explicit STO/RCL and the solve procedure in section 11.4 if the next bare key's purpose is uncertain.

The ALG **ΔFIX** prompt is an exception to immediate mode switching: the first mode-button press cancels that prompt and leaves ALG selected. Press the requested mode again to switch.

The examples use **CLEAR** on orange $1/x$ in RPN and the unshifted editing key in ALG. Prepare their stated precision, display, and stack settings with the procedures in section 3.1 and chapter 5; no example depends on old memory contents. Angle units, date order, and payment timing do not affect the example results in this chapter.

Continue two independent unfinished numbers

Use real mode, 600 working digits, FIX 4, a fixed RPN stack of 5, and no open tool or pending shift or command. Leave both numbers unfinished: do not press ENTER in RPN or equals in ALG until instructed below. Select RPN and press:

f **CLEAR** 7.00

Select ALG and enter:

CLEAR 2 + 3.0

Return to RPN. The unfinished number is still 7.00. Press 5 **ENTER**; it becomes 7.0050. Return to ALG and press 4 =. Its second operand becomes 3.04, so the result is 5.0400.
Neither mode supplied a value to the other. Both results follow from continuing the characters already present in that workspace. No memory bank is changed by this example.

Complex mode and representation changes require extra care. Unlike an ordinary RPN/ALG switch, they validate and commit pending numeric entries in both workspaces before changing the shared setting. They do not evaluate a whole ALG expression. If an unfinished ALG exponent prevents the change while RPN

is selected, return to ALG and complete, correct, or clear that entry. Section 8.1 explains this case.

14.3 Resuming an unfinished entry

After reopening, check these items before typing:

1. Read any saved-state error first. Follow section 14.9 if the message concerns saving or restoring memory.
2. Check the selected RPN/ALG button, any open tool, and the flags for a shift or pending command. A digit may be a register selector or FIX count rather than the next digit of your number.
3. Read **ENTRY** and the displayed literal. In RPN, ENTRY means a number is being entered. In ALG it means the whole expression is unfinished, even when no individual number is being typed.
4. Continue the intended entry, cancel the pending command, or deliberately clear the appropriate working area. Closing and reopening does none of those things for you.

Trailing zeros, a trailing decimal point, an exponent marker and its sign, and the selected complex component can all be resumed. Once an entry has been committed, new digits follow the normal next-operand rules rather than appending to the formatted result. In RPN the reserved stack lift has already happened before a saved partial number reappears; continuing its digits does not lift again.

Leave and resume an exponent

Use RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no pending prefix or open tool. Press:

f **CLEAR** `12.30` **EEX** +/−

The unfinished literal reads `12.30 E-`. Close the application and reopen it. With the same literal and ENTRY restored, press `2` **ENTER**. The number is $12.30 \times 10^{-2} = 0.123$, displayed as `0.1230`. The exponent-sign key did not negate the coefficient.

If a tool panel reopens, its fields may contain an unfinished or invalid draft. Edit those fields and Apply, or Close without applying. Entering digits in a field does not enter a calculator operand. A validation message can also be restored with the panel. Reopening is not an Apply action; the proposed settings have not taken effect merely because their draft fields were saved.

14.4 Cancel the pending operation before editing

A pending prefix temporarily changes the meaning of the next key. Use its indicator, not the large number alone, to decide what to press. The header's **CANCEL** dismisses any pending shift or command in one press, including CPT or a command and shift selected together. It preserves unfinished numeric entry, the ALG expression, all values, and settings, and dismisses a displayed error. CANCEL also appears while an ALG worksheet is active. An unfinished number or expression alone does not make it appear. Its position remains fixed beside the branding even when CAL is visible.

An ALG worksheet has its own CANCEL, also available through physical Escape. Worksheet cancellation returns to the suspended main expression and discards that worksheet's unaccepted drafts, not earlier accepted fields, ANS updates, or completed operations. If you relaunch before canceling, the active worksheet and its exact draft resume. Review accepted financial fields carefully: their ENTER action already stored the corresponding bank register. See section 4.12.

Table 14.2 lists other cancellation methods. CANCEL works for every shift or command in the table, including a notation digit selector; use a tool's Close control for an open tool or the worksheet's CANCEL control for an open worksheet.

Table 14.2: Canceling a pending control.

What is showing	How to cancel
Only **f** or **g**	Press that same shift again. The other shift selects the other layer instead.
STO, an arithmetic STO, **RCL**, or ALG **FUNC** or **CPT**	With no shift active, press unshifted backspace. It cancels without deleting a character. If a shift is also active, repeat that shift first to remove it, then press backspace.
HYP or **INVHYP**	Press unshifted backspace. Pressing a shift instead cancels HYP and activates that shift; a following key can then execute its alternate function.
MCLR	Press backspace or either shift once. The press cancels MCLR and is consumed, leaving no shift active; no bank, value, or unfinished entry is cleared.
DIGITS notation selector (legacy restored-state prompt)	Press unshifted backspace to cancel the selection without changing display settings.
ALG **ΔFIX**	Press any key other than a digit or equals; backspace is convenient. It cancels the whole count prompt and does not delete a count digit or edit the expression. A mode-button press also cancels without switching.

What is showing	How to cancel
ALG worksheet	Its CANCEL button or physical Escape closes it and discards unaccepted drafts; earlier accepted fields, ANS updates, and completed bank operations remain.
Open tool	Press Close. Physical Escape is the panel's Close shortcut; it is not sent to the calculator entry engine.
Full-stack inspector	Use the collapse triangle, tap outside the inspector, or press physical Escape. Calculator keyboard input is held back while inspecting.

Outside a tool, worksheet, or inspector, physical Escape cancels a pending shift, command, or displayed error without clearing a value. With none of those present, it performs CLx instead. Do not press Escape repeatedly as a general acknowledgment: the next press may clear the operand. Selecting the current entry-mode button is another way to cancel ordinary prefixes while retaining the workspace, subject to the FIX rule above.

Cancel STO and keep typing the same number

Use RPN, real mode, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Press:

f CLEAR 42 STO ← 3 ENTER

Backspace cancels STO without deleting 2 or committing 42. The next digit extends the literal to 423, and ENTER displays `423.0000`. No ordinary register is selected or changed. Without the cancellation, digit 3 would have selected register R3.

CLx or CLEAR can also cancel an ordinary pending STO/RCL, and ALG FUNC, instead of performing their clear. After cancellation, repeat the intended clearing sequence if you still want to erase the operand or workspace. Use CANCEL whenever you want to dismiss the selection without considering shift routing or the clear key's other functions.

14.5 Choose the smallest clear that fits

These clears assume that no shift or command is pending. Backspace edits the current entry, CLx clears one operand, and CLEAR clears the active working area. None is an ordinary-memory reset. For the RPN rows below, X is S0 and Y is S1; Last X and Last Y are separately saved previous operands.

Table 14.3: Entry and workspace clearing, after canceling prefixes.

Action	Effect
RPN backspace	During a real numeric entry, remove the last character. Removing its last character leaves zero in the reserved X position; it does not reverse the initial lift. With no numeric entry, backspace acts as CLx.
ALG backspace	Remove the last numeric character or, after that number is committed, the last expression element. After a completed result, clear the working expression while preserving ANS.
CLx: blue backspace in either mode	In RPN, finish an active number, save its value in Last X, and replace X with zero; preserve Y, deeper values, and Last Y. In ALG, remove the current number or trailing operand, including a closed group, while preserving the earlier expression and ANS.
RPN CLEAR: orange $1/x$	Discard the unfinished entry and zero the stack. Keep the configured fixed depth, or one zero X for unlimited depth. Preserve both Last registers, all memory banks, settings, and the ALG workspace.
ALG CLEAR: main editing key	Discard the entire expression and unfinished number. Preserve ANS, all memory banks, settings, and the RPN workspace.

In a complex entry, backspace edits the selected component; clearing all its characters leaves that component zero within the pair. CLx acts on the whole operand. To change just one accepted component, use Re/Im as described in section 8.2.

CLx is not DROP. After CLx in RPN, typing a new number lifts the zero into Y, moving an earlier Y deeper. To replace an accepted X while retaining Y as the other operand, use orange DROP on the roll-down key, then type the replacement. The complete correction examples in section 3.8 demonstrate this distinction.

Clear a memory bank deliberately

There is no one-key factory reset. MCLR is a selected group clear, not a reset of every bank, both workspaces, and all preferences. Cleared banks cannot be reconstructed with LAST X/Y or ANS. Clear only the data that the new task is intended to replace.

The sequences in table 14.4 clear entire groups, not individual register addresses. Start with no pending prefix and press each sequence as shown. For the full group-selection and cancellation rules, see section 6.10.

Table 14.4: Group clears, starting with no pending prefix.

Sequence	Scope
g STO 0	All ordinary registers and associative pairs, in either entry mode.
g STO Σ+	All statistical accumulators; RPN only.
g STO n	All five financial registers and all cash flows, in either entry mode.
g STO R↓	The RPN stack and unfinished RPN entry; both Last registers remain.

The three memory-bank clears preserve an unfinished numeric entry, expression, and stack; they do not first commit the entry. Their flags disappear because the entries are removed. A stored zero also counts as occupied, so overwriting a value with zero is not equivalent to emptying its bank. Group clearing does not reset settings or BEGIN/END.

Dedicated RPN clears offer narrower choices. Orange Sigma-plus, CL_{Σ}, removes statistics; orange FV, CL_{fin}, removes only financial registers; blue FV, CL_{cf}, removes cash flows and stores $n = 0$. These commands first finish an active numeric entry. CLcf therefore turns off CF but leaves FIN present. CLfin can leave cash flows whose selected n must be set again before use. None of these clears removes an earlier ordinary-memory copy of a statistical total.

In ALG, orange FV and blue FV have those same bank-clear scopes while preserving the main expression. For statistics, open STAT, select Clear statistics, and press CLEAR STATISTICS. This removes the bank and updates ANS to zero; the main expression and ordinary copies in R4–R9 remain. Worksheet CANCEL afterward closes the worksheet without undoing that completed clear.

14.6 Prepare a fresh problem without resetting the application

Do not toggle BGN, CMPLX, or a representation setting blindly. Read the current flag or labels first, then change only what differs from the intended starting state. A fresh problem is a known set of relevant inputs, not necessarily an empty calculator.

First close any tool or inspector, cancel pending input, and select the intended entry mode. Set the required working precision, notation, angle unit, and complex setting. Then clear the working area and initialize the inputs that belong to the task:

- **Arithmetic or a new ALG expression:** clear the active stack or expression. Retained memory flags need not be removed when the calculation does not use those banks.
- **Statistics:** clear statistics before entering a new data set. Stack CLEAR alone leaves all earlier observations in the accumulators.
- **TVM:** clear or explicitly replace the financial inputs and check BGN. Set each of the four quantities used by the target solve, including deliberate zeros. CLEAR does not remove an old PMT or FV.
- **NPV/IRR:** CF0 starts a new list. Include zero-flow periods, set the periodic rate or guess explicitly, and check n before evaluating a saved list.
- **Other financial tools:** explicitly store their required inputs. AMORT uses elapsed payments in n; simple interest uses days; depreciation uses useful life. The same register names do not imply the same input meanings.

14.7 Diagnose an unexpected result

Locate the first step that differs from the intended calculation. Compare an actual stored value with an actual stored value, or compare rounded displays at the same notation and digit count. If you compare another calculator, also match its entry, financial, date, and rounding conventions; a matching function name is not enough.

Table 14.5: Checks for a surprising result or key response.

Symptom	Check first
A digit does not enter a number	Look for STO/RCL, MCLR, a shift, or the ALG FIX prompt. Close a tool or stack inspector; tap the display to restore physical-keyboard focus.
Scientific or financial keys are missing	Show the appropriate panel. In ALG, STAT and associative legends open worksheets; financial stores and explicit CPT use different sequences from RPN.
A trig result has the wrong size	Check DEG/RAD/GRAD and the unit of each ALG function already entered. Changing the setting does not convert an operand. See section 7.4.
Subtraction, division, a power, or root is reversed	RPN uses Y (S1) before X (S0). Inspect the stack without rolling it; use SWAP if the operands are reversed. In ALG check grouping and precedence.
Extra operands remain	ENTER during entry confirms; idle ENTER duplicates. Percentage functions retain Y; statistical and financial commands have their own result counts and lift rules.

Symptom	Check first
An ALG total seems unchanged	The large display can be the current operand or a preview. Read the expression and press equals to evaluate it. Repeated equals does not repeat an operation.
Digits differ, or a small value shows an exponent	Read the pinned exponent and notation count. FIX can use scientific fallback. Display rounding does not round stored values; increasing working precision does not repair old approximations. See chapter 5.
A component, sign, or complex root differs	Check COMPLEX, Re/Im versus polar labels, component selection, and principal branches. Complex mode off hides existing imaginary parts; new stores, recalls, and calculations in that mode write real values. See chapter 8.
A memory recall is zero or the wrong value	Check bank and address. Unset ordinary registers and unmatched associative keys return zero. Rounded-looking real keys may differ internally. Associative keys with the same real part address one entry in either mode; a later store replaces it. Real-mode recall copies only the real component, leaving the source memory unchanged.
A statistical answer includes old data	Check the initial statistics clear, RPN's y-before-x entry order or ALG's labeled X/Y fields, count, corrections, and the difference between a weight and repeated samples. See chapter 10.
A financial key stores instead of solving	In ALG, a bare register key always stores; use CPT then the target register to solve. In RPN, recall, ENTER, mode selection, and ordinary arithmetic affect bare-key readiness. Use the procedure in section 11.4.
A loan or savings answer differs	Recall all four non-target inputs. Check old PMT/FV, cash signs, n and i period units, BGN, fractional-period conventions, and deliberate payment rounding.
NPV or IRR differs	Review the expanded list, omitted zero periods, signs, n selection, periodic i, and IRR guess/bounds. IRR does not automatically replace i. See chapter 12.
Dates, bonds, or depreciation differ	Check MDY/DMY, six fractional date digits, day-count/coupon conventions, clean price, salvage, and the meaning of n/i for that tool. See chapter 13.
Finish the expression first	A direct ALG financial calculation would replace unfinished compound work. Complete it with equals or clear it; storage still uses the displayed operand.
A worksheet action rejects edited fields	ENTER accepts a selected field; equals only evaluates it. Accept every edited field before computing. CANCEL closes the worksheet and discards its unaccepted edits; reopen it to compute from retained accepted inputs.

Symptom	Check first
A conversion has the wrong scale or offset	Check Dimension, From, To, US/Imperial names, and temperature reading versus interval. Apply uses the current operand, not an unfinished ALG total. See chapter 9.

Some limitations are properties of the numerical methods rather than entry mistakes. Custom complex special-function algorithms and raw-sum statistical formulas can lose accuracy through cancellation. The relevant subject chapters describe their numerical limits; a longer readout alone cannot resolve them.

14.8 Recover from a calculation error

A calculation error is different from a storage failure. An operand may be outside a function's domain, an expression may be incomplete, or a solver may be unable to find a result. Read the exact message before clearing it. The next enabled calculator key normally clears the message *and performs that key's action*; it is not merely an acknowledgment key. Physical Escape can dismiss an error without performing arithmetic, provided that a message or prefix is still active.

An attempted operation can finish numeric entry before detecting its error. The retained operand can therefore be correct even though **ENTRY** has changed. In RPN, check X (S0) and Y (S1) before backspacing: with entry no longer active, backspace is CLx. In ALG, a committed number is an expression element, so backspace can remove it as a whole rather than remove its final digit. A failed ALG evaluation retains the expression and does not replace ANS.

Correct the sign after a rejected factorial

Use RPN, real mode, fixed depth 5, 600 working digits, FIX 4, and no pending prefix. Show the Scientific panel and press:

f CLEAR 2 +/− $x!$

Real factorial rejects the negative argument. The entry is now committed: X remains `-2.0000`, with ENTRY off and an error message. If the intended argument was positive 2, press:

+/− $x!$

Sign change clears the message and negates the retained operand; factorial returns `2.0000`. A physical Escape once before this correction would only dismiss the error. No operand recall is needed, because the rejected factorial did not replace X with an answer.

Finish an ALG group after evaluation fails

Use ALG, real mode, 600 working digits, FIX 4, and no pending prefix. Press:

CLEAR 2 + (3 × 4 =

The message reports unbalanced parentheses. The expression remains available; press) =. The closing parenthesis is processed while the previous message is dismissed, and evaluation returns `14.0000`. The calculation is $2 + (3 \times 4)$, not a request to start again from the displayed 4.

Interpret the kind of failure

For missing operands, supply the required input or replace the incomplete expression. For a domain error, check signs, integer requirements, zero divisors, and whether the selected real or complex function is appropriate. Statistics may need more observations or nonzero variance; financial tools may require explicitly stored registers even when a recall of an unset register would show zero. These are input conditions, not requests for more display digits.

A convergence or bracketing error describes an unsuccessful numerical search. Check the model, scale, and any IRR bounds; it does not by itself prove that no mathematical root exists. A capacity or expression-limit message requires a smaller problem or a different decomposition. Unlimited RPN still needs enough actual operands, and cyclic roll is unavailable with an unlimited stack.

RPN can also display `NaN`, ∞, or $-\infty$ from basic arithmetic without a separate rejected-command message. NaN means not a number. Treat these as nonfinite

values, not ordinary results. Check the function's domain and replace or clear the affected operand before continuing. ALG rejects nonfinite expression results. Complex mode extends some domains but does not make division by zero or every singular function finite.

For Σ+ or Σ−, the message Statistics samples must be finite means a sample's real component is NaN or infinity. Statistics totals must be finite means an existing total is nonfinite or the proposed update would produce one. Both errors preserve the statistical bank; no sample is partly added or removed. Correct the operands and retry, or clear and re-enter the observations if the saved bank already contains nonfinite totals. See chapter 10.

The warning Used real portion only means that a real-only operation omitted nonzero imaginary data while complex mode was active. It can accompany a successfully computed real answer; it is not a save failure. Later input can remove the warning, so its absence does not certify that all stored inputs were real. See section 8.7.

LAST X and LAST Y can help after a *successful but unintended* RPN operation. They recall saved previous operands, distinct from the current X (S0) and Y (S1); recover them before another command overwrites the saved values. They do not provide a general error rollback or restore an earlier memory bank. Use the worked recovery sequence in section 3.8. If the original inputs are uncertain, record what remains, then re-enter a known problem rather than applying a series of guessed reversals.

14.9 Respond to a saved-state error

The messages in this section concern the local database, not the mathematical validity of a calculation. Record the full message before trying to continue. A long message can include a specific SQLite operation, error code, or state-validation detail that distinguishes an access problem from incompatible or damaged saved data.

Unable to save; the last saved state was restored

The application tried to save an action and could not finish the transaction. It then successfully reloaded the last durable calculator and presentation state. The attempted action has not been accepted as new saved work. The screen can return to an earlier entry or pending command, not merely an earlier displayed number.

For example, if the saved state was 42 followed by STO, and selecting R3 could not be saved, the restored state can again show the editable 42 and the pending STO. The register selection has not completed. Read those restored flags before

deciding whether to retry the selector or cancel storage; typing a new calculation immediately could instead complete the pending command.

Check any cause identified by the message, such as an unavailable or unwritable storage location, and resolve that condition before relying on another save. After it is resolved, retry the intended action from the state actually restored. There is no separate Retry button. If the message recurs, stop making changes, record the failure and visible state, and report it. Repeated clears are also writes and do not repair a database access problem.

Restore failure or unavailable calculator memory

The message Unable to restore calculator memory appears when startup cannot open or reconstruct the saved calculator. Calculator memory is unavailable can appear when an unsuccessful save is followed by an unsuccessful attempt to reload it. The application disables calculator keys instead of silently replacing the saved database with a blank one. A zero or an older number still visible on the screen is not proof that the saved banks are empty or usable.

There is no editable recovery workspace, database-repair control, or factory-reset button in this screen. Record the full message and the application version, then obtain help with that particular storage failure. If the message says that the state was saved by a newer application, use the compatible build that created it rather than assuming that an older binary can read it. Reopening can retry startup, but cannot by itself repair invalid data or make an incompatible format compatible.

Preserve the affected installation's data while investigating. Deleting the application data, removing its database, or creating a replacement database discards the saved work; those actions are not the same as canceling an entry and are not a recovery procedure supplied by this interface. Do not treat reinstalling as an assured way to preserve or repair calculator memory.

Information for a source-build investigation

The normal database is `Calculator.sqlite` in the application's `SpicyRPN` directory under Application Support, within its sandbox. The exact enclosing path depends on the platform and installation; it is not a document you select from the calculator. The application uses SQLite's WAL journal mode with synchronous FULL for its saves. A developer investigating a copy must preserve the database and any associated WAL state consistently; copying only an arbitrary live main file is not an application backup command.

Xcode previews and builds launched with the developer's ephemeral option intentionally use in-memory state and do not retain it across process termination. Relaunch tests can use a separate temporary test database. If work seems to vanish only

in a development or test run, include the launch configuration in the report. See chapter D for the supported build and test procedures.

14.10 Identify the build and report a reproducible problem

This manual identifies application version 2.0.0 (build 3). Open **HELP**, choose Licenses & Acknowledgments, then Credits to read the installed version and build. For a source build, the app target records MARKETING_VERSION and CURRENT_PROJECT_VERSION; the generated bundle records CFBundleShortVersionString and CFBundleVersion. Also record the source package version and any local changes when reporting a problem with your own build.

A useful report gives a short sequence that another person can follow without inheriting your current calculator. Include:

- Application version and build; source package version and local changes, if applicable; device or Mac model and OS version; screen orientation or window size.
- RPN or ALG; real or complex mode and representation; working and display digits; notation, angle unit, stack depth, date order, and payment timing wherever they affect the example.
- A complete preparation: which workspace or bank was cleared, which memory values or financial inputs were stored, and whether any entry, shift, or prefix was active.
- The exact key sequence, including shifts, panel changes, tool or worksheet field names and values, and the Apply, ENTER, COMPUTE, Use value, Close, or CANCEL action. Identify physical shortcuts when those were used instead of on-screen keys.
- The expected value or behavior and its basis, the actual result with exponent and relevant flags, and the first differing step. Include the whole error or warning text.
- Whether it repeats from that preparation and whether reopening changes it. For a persistence problem, identify the final successful action, the failing action, and what returned after relaunch.

A screenshot can document a layout defect or exact flag arrangement, but it does not show every hidden register or undisplayed digit. Include the relevant input values and the sequence in text as well. For financial work, state the period and rate conventions; for dates, write the intended calendar dates alongside their numeric encodings. For a numerical comparison, distinguish a difference in rounded output from a difference in retained value or algorithm.

If the report requires sensitive amounts, replace them with a small reproducible example when possible. A saved database can contain both current values and

their calculation provenance, so do not assume it is equivalent to a screenshot of the display. Use the project's report channel with the information needed to reproduce the problem, and retain the original data separately while the issue is investigated.

Chapter 15

Complete Command and Settings Reference

Use this chapter to find a key, check operand order, or determine what a command changes. The keyboard maps include every main and alternate legend. The alphabetical entries group closely related operations while identifying their individual formulas and restrictions. The explanatory chapters provide longer examples; chapter C supplies the numerical conventions behind the results.

This reference describes the exposed calculator controls. Both entry modes support scientific, financial, statistical, and memory operations. ALG uses explicit CPT and labeled worksheets where RPN uses stack operands; the mode-specific procedure matters.

15.1 Reading an entry and following an example

A key shown after f or g identifies the physical main key to press, unless its alternate legend is explicitly named. Thus f PMT selects AMORT; g PMT selects RND. The INT legends have two meanings: orange ENTER in RPN and orange $1/x$ in ALG truncate a number, while blue financial i selects simple interest.

The following effect codes make the mathematical entries compact. They describe a successful RPN operation after an active number is committed. ALG uses the expression rules immediately below instead. Unless an entry says otherwise, a scientific function changes no memory bank or setting.

In RPN instructions, X is the current value in the visible S0 position; Y is the next value in S1. These are current stack positions. Last X and Last Y are separate saved operands from earlier operations, not other names for S0 and S1. Statistical coordinates and ALG fields labeled X and Y identify inputs; their labels do not imply RPN stack positions.

Table 15.1: RPN effect codes used in the alphabetical reference.

Code	Result and saved operands
U	Unary: replace X with $f(X)$, preserving deeper values. Save old X in Last X and old Y in Last Y if Y exists.
B	Binary: calculate $f(Y, X)$, consume both operands, and return one result in X. Save both operands in Last X/Y; drop the stack once.
P	Percentage: calculate $f(Y, X)$ but replace only X. Preserve Y and deeper values; save both original operands in Last X/Y.
Z	Zero operands: lift a new value into X, preserving both Last registers.

A fixed stack discards its deepest value on a lift and repeats its previous deepest value on an ordinary drop. An unlimited stack grows and shrinks, keeping one zero if DROP empties it. A binary function requires two positions. A unary function on a one-position unlimited stack preserves Last Y. For financial and statistical entries, the detailed stack procedure is RPN-specific; each entry also gives its ALG route separately. Special statistical and financial effects are stated separately; they must not be inferred from the number of inputs. With CMPLX off, each newly written value and each newly saved Last operand has an imaginary component of zero. Moving an existing stack value preserves its components. See chapter 2 and section 6.5.

In ALG, a unary function opens a call when an operand is expected; after an operand it applies to that preceding number, group, or function result. A binary key goes between its first and second operands. FUNC followed by a unary or binary key instead opens an explicit call; separate two arguments with g FUNC and close with). The first argument corresponds to Y and the second to X. Constants insert a factor after an existing operand. ANS and recall instead replace a trailing operand. Successful equals, worksheet ENTER, and direct or worksheet computation update ANS. Worksheet calculations use a declared primary result; browsing outputs does not change it. These operations never alter the suspended RPN stack or its Last registers. See chapter 4.

The reference start

Every numerical example below starts independently in RPN, real mode, fixed depth 5, 600 working digits, and FIX 4, with no open tool, active shift, or pending command. Before its displayed sequence, press f $1/x$ for CLEAR. The setup procedure is in section 2.2. Where an entry specifies ALG, select ALG, retain real mode, 600 digits and FIX 4, and start with its unshifted CLEAR instead. Examples involving angles specify DEG; date examples specify MDY and FIX 6. Results usually omit trailing display zeros. A negative input is entered as its magnitude followed by +/−.

No example assumes an empty memory bank. Examples that store a register, clear statistics, or start cash flows deliberately replace that data. Financial examples initialize every input they use. A semicolon in a printed sequence only separates steps for readability; it is not a key.

15.2 Complete keyboard maps

Read each map top to bottom, with a rule separating physical rows where useful. A dash means there is no alternate binding; selecting that layer does not execute the main function. In portrait, swipe the top two rows left for Financial or right for Scientific. Paging preserves entry, shifts, and pending commands. In landscape, both panels are visible. The remaining rows stay fixed. RPN ENTER occupies two row heights. ALG's left entry column has normal-size EEX, $1/x$, +/−, and equals keys from top to bottom, placing equals on the bottom row. RPN has 49 key positions, with 39 visible in portrait; ALG has 50, with 40 visible in portrait. See section 1.5 for the arrangement.

Table 15.2: Scientific panel: both entry modes.

Main key	Orange f	Blue g
SIN	DEG	SIN^{-1}
COS	RAD	COS^{-1}
TAN	GRAD	TAN^{-1}
HYP	HYP^{-1}	atan2
Γ	$\ln\Gamma$	nPr
$\sqrt{x}$	x^2	erf
e^x	LN	erfc
10^x	LOG	LOG_2
y^x	$\sqrt[x]{y}$	$\log_y x$
$x!$	$\ln x!$	nCr

For direct ALG computation, first finish or clear any unfinished compound expression; a lone scalar is allowed. In ALG, the five financial register keys always store the displayed operand. CPT then a register solves it; RCL then a register recalls it. CPT followed by NPV or IRR computes directly from the bank. Unprefixed cash-flow and multi-input financial keys open labeled worksheets. BEGIN/END, date order, CLfin, and CLcf retain their shared settings or clear scopes. Orange n/i and RND are direct calculations. See sections 4.11 and 4.12.

The shift keys themselves select a layer regardless of the current layer; pressing the same shift twice cancels it. Their map cells describe that control, rather than additional shifted commands. The ALG middle rows have no stack, roll, DROP, or

Table 15.3: Financial panel: both modes, with different entry procedures.

Main key	Orange f	Blue g
n	12×	BGN toggle
i	12÷	INT (simple interest)
PMT	AMORT	RND
PV	PRC	YTM
FV	CL_{fin}	CL_{cf}
CF_0	DATE	ΔDAYS
CF_j	MDY	DMY
N_j	SL	SYD
NPV	DB	DB→SL
IRR	IRR[a,b]	SHOW

Table 15.4: The two fixed middle rows in RPN.

Main key	Orange f	Blue g
Σ+	CL_{Σ}	Σ−
$\bar{x}$	Weighted mean, $\overline{xw}$	SD
L.R.	Estimate $\hat{y}$ and correlation	RCL_{Σ}
STO	Associative STO_{xy}	MCLR
RCL	Associative RCL_{x}	HELP
f	Select/cancel orange layer	—
g	—	Select/cancel blue layer
R↓	DROP	R↑
$x \leftrightarrow y$	STACK	LAST X
$1/x$	CLEAR	LAST Y

Table 15.5: ALG Expression row, followed by Shift and Memory.

Main key	Orange f	Blue g
(	—	—
)	—	—
FUNC	STAT	Comma
CPT	—	—
CLEAR	—	—
f	Select/cancel orange layer	—
g	—	Select/cancel blue layer
ANS	—	—
STO	STO_{xy}	MCLR
RCL	RCL_{x}	HELP

LAST keys. CPT starts explicit computation; STAT and the associative-memory legends open worksheets. See sections 4.11 and 4.12.

Table 15.6: Entry and arithmetic keys; mode-specific keys are marked.

Main key	Orange f	Blue g
EEX	—	—
7	FIX	PREC
8	FSE	CONV
9	DRG	NINT
÷	CMPLX	MOD
ENTER (RPN, two rows)	INT (integer part)	FRAC
$1/x$ (ALG)	INT (integer part)	FRAC
4	Re/Im	RAN#
5	P↔R	Ceiling, $\lceil x \rceil$
6	Re↔Im	Floor, $\lfloor x \rfloor$
×	ABS	REM
+/− (ALG)	—	—
1	—	%
2	—	Δ%
3	—	T%
−	—	—
+/− (RPN)	—	—
= (ALG)	—	—
0	π	—
.	Euler's e	—

Main key	Orange f	Blue g
←	ln 2	CLx
+	—	GOAT

FIX uses a different input sequence in each mode. SCI and ENG have no direct current key: use FSE to cycle to them. Orange 1, 2, and 3 have no binding. CONV, PREC, STACK, IRR[a,b], SHOW, and HELP open panels; their fields are separate from calculator operands.

15.3 Physical-keyboard shortcuts

Give the calculator display input focus before typing. These shortcuts select calculator keys; they are not a text field for pasted formulas or function names. Unless the table states otherwise, an active f or g layer changes the selected key in exactly the same way as a tap.

Table 15.7: Complete calculator-character shortcuts.

Physical key	Action
0–9	Corresponding digit key.
.	Decimal-point key.
,	RPN: decimal-point key, subject to an active shift. ALG: force the blue FUNC argument separator.
+	Addition key.
- or −	Subtraction key; not change sign.
* or ×	Multiplication key.
/ or ÷	Division key.
^	Power key, y^x.
e, E	EEX; not Euler's constant.
s, S	Change-sign key.
f, F	Orange shift.
g, G	Blue shift.
p, P	Force orange 0 to insert π. Unavailable while a command prefix is pending.
Return, newline, =	ENTER in RPN; equals in ALG.
Backspace/Delete	Calculator backspace.
(,)	Parentheses in ALG; no corresponding RPN key.
Escape	Cancel active shift, pending command, or error first. With none, perform CLx. In the full-stack inspector, close the inspector instead.

A parameter panel receives keyboard input in its own fields, and the stack inspector holds back calculator input while open. With the ALG FIX prompt active, digits collect its count and Return or equals accepts it; another keyboard character cancels the prompt without entering that character into the expression. No dedicated shortcut types an entire scientific-function name.

15.4 Alphabetical reference: A through F

ABS — absolute value

Keys/modes: f × , both modes; **effect U**. Return $|X|$ in real mode, or the real magnitude $\sqrt{a^2+b^2}$ of $a+bi$ in complex mode. Angle and display settings do not change its mathematical meaning. Example from the reference start: 3 +/− f × gives 3. See section 7.6 and chapter 8.

AMORT — a block of equal payments

RPN keys and effect: f PMT , RPN. X supplies an additional whole payment count, 0–100 000. Stored n is elapsed payments, i is periodic percent, PV the signed balance, and PMT the oppositely signed nonzero payment; all four must be explicitly stored. FV is unused. Require nonnegative integer n and $i > -100$. An unlimited stack needs two operands even though old Y is not an input to the financial formula. Consume X and return total interest in X, total principal in Y, moving old Y to S2. Save old X/Y; update financial n and PV. BEGIN waives interest only on the first payment when initial n is zero. Intermediate amounts are not rounded to the display count.

Example, reference start with END selected: enter 0 STO n ; 1 STO i ; 1000 STO PV ; 100 +/− STO PMT ; 2 f PMT . Results are X = −19.1, Y = −180.9, n = 2, PV = 819.1. See section 11.11.

In ALG, the same key opens Amortization; accepted named inputs produce Interest, Principal, Balance, and updated n. See section 11.12.

ANS — previous ALG answer

Keys/modes: ANS , ALG only. Insert the last successful ALG result at an operand position, or replace the trailing operand or closed group. After a completed result it starts a new expression. It does not evaluate the expression or change ANS itself. CLEAR and failed evaluation preserve ANS; its initial value is zero. With CMPLX off, insertion copies only the real component and leaves the saved answer intact. Example from the ALG reference start: 2 + 3 = ; CLEAR 10 ÷ ANS = gives 2. See section 4.8.

Arithmetic — add, subtract, multiply, divide

Keys/modes: [+], [−], [×], [÷], both modes; **effect B**. Calculate $Y + X$, $Y - X$, YX, or Y/X, using both complex components when enabled. Division needs a nonzero divisor for a finite answer. ALG gives multiplication/division priority over addition/subtraction; equal-priority chains run left to right. Examples, each from the reference start: 12 [ENTER] 3 followed by the respective operator gives 15, 9, 36, or 4. A leading ALG minus has lower priority than a following power. See sections 3.3 and 4.2.

Associative storage and recall

Keys/modes: [f] [STO] and [f] [RCL], both modes. In ALG these open Key/Value worksheets; accept inputs and choose STORE or RECALL, then Use value for insertion. See section 6.7. In RPN, store uses key X and value Y, saves both Last operands, and drops X. Recall replaces key X with its value, saving the key in Last X and preserving Last Y. A missing key returns zero without storing a pair. Numeric equality of the real components determines a match in both modes. The key is stored as a real value; an imaginary component is ignored and a NaN real component is invalid. Keys with equal real parts address the same entry, so storing again replaces its value. The value copied from Y or recalled into X includes both components with CMPLX on and only its real component with CMPLX off. Recall leaves the stored source unchanged.

Example: 42 [ENTER] 123 [f] [STO]; 123 [f] [RCL] returns 42, replacing any earlier pair for that key. The bank is distinct from R0–R9. See section 6.7.

atan2 — direction from two coordinates

Keys/modes: [g] [HYP], both modes; **effect B**, real-only. Supply vertical coordinate Y first, horizontal X second. Return the quadrant-aware angle atan2(Y, X) in the selected angle unit, within a half-turn on either side of zero. The origin has no geometric direction. Example, DEG: 1 [ENTER] 1 [+/−] [g] [HYP] gives 135 degrees. See section 7.4.

Backspace — delete the trailing character or element

Keys/modes: [←], both modes. During entry, delete the selected component's last character, removing exponent digits before its exponent marker. In real RPN, deleting the final character leaves zero in the reserved X position; it does not undo the original lift. Idle RPN backspace performs CLx and saves old X. In ALG, idle backspace removes the last expression element, or clears a completed working expression while retaining ANS. Pending-command cancellation takes

priority, including MCLR: backspace cancels it without clearing memory. Example: 12.36 ← 5 ENTER gives 12.35 in RPN. See sections 3.8 and 4.7.

BGN — BEGIN/END payment timing

Keys/modes: g n, both modes. Toggle beginning-of-period payments; BGN present means BEGIN, absent means END. In RPN, accept active entry without moving the committed stack or changing the Last registers; preserve TVM store/solve readiness. ALG preserves its main expression. Timing affects TVM and AMORT, not explicit cash-flow dates, bonds, or depreciation. Example: with BGN absent, this key sequence shows BGN; repeating it restores END. Bank clears preserve the setting. See section 11.3.

CAL — digit ruler

Keys/modes: header CAL, both modes when a number overflows its display. Toggle the ruler that marks coefficient digits in groups of five. It neither changes the value nor commits entry, and it preserves any pending shift or command. Example: open PREC, set display digits to 100, apply, then insert π with f 0; use CAL and horizontal scrolling to inspect the long result. The ruler is a position aid, not an accuracy indicator. See section 5.5.

CANCEL — dismiss a pending shift or command

Keys/modes: header CANCEL, both modes. Appears while an ALG worksheet is active or a shift or command is waiting for another key: f, g, STO (including arithmetic modifiers), RCL, MCLR, HYP/INVHYP, FUNC, CPT, a notation digit selector, or the ALG FIX prompt. Cancels the whole selection, including any associated shift, and dismisses a displayed error. Does not commit or edit a number, evaluate an expression, clear a value or bank, or change settings. The cancellation is saved. Numeric entry or an unfinished expression alone does not show this button. Its fixed position beside the branding does not depend on CAL visibility. Example: 42 STO CANCEL 3 continues the same entry as 423 without storing it. An ALG worksheet provides its own CANCEL and physical Escape shortcut. Canceling the worksheet discards its unaccepted field edits and closes it; earlier accepted values, ANS updates, and completed operations remain. See sections 1.8, 4.12 and 14.4.

Cash-flow entry — CF0, CFj, Nj

RPN keys and effect: CF_0, CF_j, N_j, RPN. CF0 replaces the whole list with X and stores n = 0. CFj appends X to an existing list and stores its final index in n. Nj takes an integer count 1–1000 and appends count-minus-one copies of the actual

last flow, then updates n. Each accepts entry, preserves the committed stack and both Last registers, and requires the expanded list to fit 1000 entries including CF0. CF0 and CFj accept finite signed amounts, including zero. A repeated amount has separate period indices, not an editable group frequency.

Example: 100 +/− CF_0; 30 CF_j; 3 N_j creates $(-100, 30, 30, 30)$, with n = 3 and X still the frequency 3. See section 12.3.

In ALG, these keys open initial-flow, next-flow, and frequency worksheets. Accept fields and choose STORE, APPEND, or REPEAT; see section 12.9.

Cash-flow indexed STO and RCL

RPN keys and effect: STO CF_j or RCL CF_j, RPN; CFj is unshifted. An index in X must identify an existing entry, starting at 0. Recall replaces it with the flow and saves the index in Last X. Store uses amount Y and index X, drops the index, and leaves the amount in X; it also saves only Last X. Last Y, n, and list length are unchanged. STO followed by an arithmetic operator before CFj applies old flow op Y; division requires nonzero Y. The adjustment remains on the stack.

Example: 100 +/− CF_0; 110 CF_j; 120 ENTER 1 STO CF_j; 1 RCL CF_j gives 120 with n still 1. There is no indexed insertion or deletion. See section 12.4.

In ALG, the same sequences open indexed worksheets with a Cash-flow index and, for storage, Value. Arithmetic STO chooses the corresponding modification. See section 12.9.

Ceiling and floor

Keys/modes: g 5 for ceiling, g 6 for floor, both modes; **effect U**, real-only. Ceiling returns the least integer at least X; floor returns the greatest integer at most X. Neither depends on display digits. Examples from separate reference starts: 2.75 +/− followed by blue 5 gives −2; blue 6 gives −3. See section 7.6.

CLEAR and CLx — working-area clears

Keys/modes: RPN CLEAR is f $1/x$; ALG has unshifted CLEAR. Both modes use g ← for CLx. RPN CLEAR discards entry, fills a fixed stack with zeros or leaves one unlimited zero, and preserves both Last registers. RPN CLx commits and saves old X in Last X, then replaces X with zero, preserving deeper values and Last Y. ALG CLEAR discards the whole expression; CLx removes current entry or the trailing operand/group. Both preserve ANS. These clears preserve all memory banks and settings. Cancel an ordinary pending prefix before clearing; otherwise the first clear only cancels that prefix.

Example: 8 ENTER 3 g ← leaves X = 0, Y = 8 in RPN; then f $1/x$ zeros the whole stack. See sections 2.8 and 4.9 and chapter 14.

CLcf, CLfin, and CLΣ — dedicated bank clears

Keys/modes: In both modes, g FV removes cash flows and stores n = 0; f FV removes all five financial registers but retains cash flows. RPN f Σ+ removes the six statistics accumulators; ALG provides Clear statistics under STAT. In RPN these commands commit active entry and preserve the committed stack and both Last registers. ALG preserves its expression; its CLEAR STATISTICS action updates ANS to zero. Settings are retained. CLcf therefore leaves FIN lit; CLfin can leave CF lit. Copies of statistics in R4–R9 survive CLΣ. Example: 0 CF_0 lights FIN and CF; g FV removes CF while FIN remains. Use MCLR for a group clear that preserves unfinished entry. See sections 6.10, 10.1 and 12.11.

CMPLX and complex component controls

Keys/modes: both modes. Finish any incomplete ALG number before changing complex mode or representation, even if ALG is the inactive workspace. f ÷ toggles complex mode. With complex mode enabled, f 4 selects the other component; f 5 toggles rectangular/polar entry and presentation; f 6 swaps the rectangular real and imaginary parts. Mode and representation changes commit pending numbers in both workspaces without evaluating an ALG expression.

Re/Im stays within one operand; on an accepted value it reopens the second component, and a new digit replaces that component. In ALG, a closed group or function call is replaced by its edited value; the surrounding expression and ANS are preserved until evaluation. Swapping accepts entry and replaces the pair in place, preserving the RPN Last registers. Polar form uses magnitude and the selected angle unit; swapping still exchanges rectangular parts. Disabling CMPLX hides existing imaginary parts but retains them until a later operation replaces the value. New calculations, stores, recalls, Last-register captures, and idle ENTER copies made with CMPLX off write zero imaginary components. A real-mode recall leaves its source memory unchanged; turning CMPLX back on does not restore a component in the newly recalled copy.

Example from the reference start: f ÷ enables complex mode; select rectangular form if needed, then enter 3 f 4 4 ENTER. The pair is $3 + 4i$. Orange 6 gives $4 + 3i$; applying it again restores $3 + 4i$. In DEG, orange 5 shows magnitude 5 and phase 53.1301 degrees. ALG keeps its expression visible and accessible alongside the two component rows. See chapter 8 for ALG component-editing examples.

Constants — pi, Euler's e, and ln 2

Keys/modes: f 0, f ., and f ←, respectively, both modes; **effect Z**. Generate the real constant at current working precision. ALG inserts an implicit product after an operand, or starts new work after a completed result. No FUNC is needed. From separate reference starts, these keys display 3.1416, 2.7183, and 0.6931 under FIX 4. They preserve both RPN Last registers. See section 7.12.

CONV — unit conversion

Keys/modes: g 8, both modes. First enter or recall the value to convert, then open CONV. Choose Dimension, From, and To, then Apply. Conversion uses the current real component, including an active number, and returns one real value. RPN replaces X with no movement and preserves both Last registers. ALG replaces the trailing operand or closed group; after a completed result, it starts a new expression. It does not update ANS until equals. Close leaves entry untouched; Apply rejects a nonfinite result. Units are not attached to the stored number.

Example: enter 12, open CONV, choose Length, inch to foot, and Apply: X becomes 1. Source memory and settings remain unchanged. The full inventory appears in section 15.10; factors, affine temperatures, and numerical caveats are in chapter 9.

CPT and STAT — ALG computation and worksheets

Keys/modes: main CPT; orange FUNC = STAT, ALG only. Finish or clear an unfinished compound expression before solving; a lone scalar is allowed. Then press CPT and an unshifted n, i, PMT, PV, FV, NPV, or IRR target. Successful computation becomes a completed result and ANS. CANCEL discards the prefix, and a saved prefix resumes without running the computation.

STAT opens the add-sample worksheet with an operation selector for sample removal, means, weighted mean, sample deviations, regression, estimation, summation recall, and statistical clear. Related finance and associative-memory commands use the same worksheet interaction. Equals evaluates a selected field; ENTER accepts it. Both update ANS. The named action uses accepted inputs and updates ANS with its primary output. Financial-register fields write their registers on acceptance. Use value inserts the selected output in the suspended main expression; output browsing does not change ANS. CANCEL discards unaccepted edits and closes; completed work remains. All worksheet inputs, exact drafts, outputs, and selections are retained across relaunch. See sections 4.12 and 10.10.

DATE and ΔDAYS

RPN keys and effect: f CF_0 adds an integer count X to date Y; g CF_0 finds the interval from date Y to date X. RPN, real-only. DATE consumes two operands and returns one encoded date. ΔDAYS replaces the two dates with actual days in X and the implemented 30/360 count in Y. Both save original X/Y, use MDY/DMY, and leave financial banks unchanged. Gregorian years are 1–9999; DATE's count must be within ±3 652 058 and its result within that calendar.

Examples, MDY selected with f CF_j and FIX 6 selected with 6 f 7: `2.282024` ENTER 2 f CF_0 gives `3.012024`. From a fresh clear, `1.312023` ENTER `3.312023` g CF_0 gives X = 59 and Y = 60. The 30/360 rule has no February-end adjustment and is not generally antisymmetric when dates are reversed. See sections 13.4 and 13.5.

In ALG, the same shifted keys open labeled date worksheets. DATE returns the result date; ΔDAYS returns Actual days and 30/360 days. See section 13.12.

DEG, RAD, GRAD, and DRG — angle settings

Keys/modes: f SIN, f COS, f TAN select degrees, radians, gradians; orange 9 cycles DEG, RAD, GRAD. Both modes. Committed values, stack positions, and Last registers stay unchanged. Unfinished entry keeps its typed digits; for a pending polar number, however, the new unit changes the meaning of the typed phase when it is committed. A committed complex value keeps its direction and is displayed in the new angle unit. The setting governs trig arguments/results and polar angles, not hyperbolic functions or the imaginary part of a logarithm. ALG trig functions retain the unit from their insertion. Example: from DEG, orange 9 selects RAD; orange SIN returns to DEG. CONV changes an angle's numerical units instead. See sections 7.4 and 8.3.

Depreciation — DB, DB to SL, SL, and SYD

RPN keys and effect: RPN: orange NPV = DB; blue NPV = DB→SL; orange Nj = SL; blue Nj = SYD. Explicitly store life n, cost PV, salvage FV, and, for DB methods, positive factor i in percent. X is integer year $j \geq 1$. Require $n > 0$ and $PV \geq FV \geq 0$; PMT is unused. Lift depreciation into X and remaining value above salvage into Y, retaining year in S2 if capacity permits. Preserve financial inputs and both Last registers. Each request recomputes from original cost; it does not advance a ledger.

SL spreads the depreciable base across life; SYD uses descending year weights. DB multiplies opening book value by $(i/100)/n$, capped at salvage, and writes off the remainder in the final year. Crossover uses the larger of DB and remaining-life SL

before that final write-off. Fractional lives have the conventions in section 13.11; active DB years are limited to 100 000. Years beyond $\lceil n \rceil$ return two zeros.

Example: 5 **STO** **n**; 1000 **STO** **PV**; 100 **STO** **FV**; 200 **STO** **i**. Clear the stack, enter 2, and select each method separately. SL returns 180 and 540; SYD 240 and 360; DB and DB→SL both return 240 and 260. These are year-2 amount and remaining depreciable value. See section 13.8.

In ALG, the same legends open worksheets for Year, Life, Cost, Salvage, and the DB factor where needed. Depreciation is primary; Book value less salvage is separately available. See section 13.12.

Digits, decimal point, EEX, and sign

Keys/modes: **0**–**9**, **.**, **EEX**, **+/−**, both modes. Digits build one number. RPN's first digit, decimal, or EEX lifts once; ENTER commits without a second lift. A new ALG number replaces a trailing accepted operand, except after a closing group where it starts an implicit product; after equals it starts a new expression. EEX introduces an integer power of ten, using mantissa 1 if it starts an ordinary new number. An empty second complex component needs an explicit mantissa. Sign edits the mantissa or active exponent; outside entry it negates the complete operand (U in RPN). ALG requires exponent digits before evaluation; RPN accepts an empty exponent as zero. Repeated decimal points or EEX markers are ignored.

Example: 4 **+/−** **EEX** **+/−** 3 **ENTER** gives −0.004. See sections 3.2, 4.1 and 8.2.

DROP — discard X

Keys/modes: **f** **R↓**, RPN. Commit entry and remove X, moving Y down. A fixed stack repeats its deepest value; unlimited shrinks, leaving zero if emptied. Preserve both Last registers and all memories. Example: 8 **ENTER** 3 **f** **R↓** returns 8. Use this before entering an accepted operand's replacement. See section 2.7.

ENTER and equals

Keys/modes: tall **ENTER** in RPN, **=** in ALG. RPN ENTER finishes active entry without duplication; idle ENTER lifts a copy of X. With CMPLX off, the new X copy is real, while the original moved to Y retains its components. Both preserve Last X/Y. ALG equals commits entry, evaluates the complete expression, and updates ANS; repeated equals leaves a completed answer unchanged and records it as ANS again. Missing parentheses are not supplied. Example in RPN: 6 **ENTER** **ENTER** **+** gives 12. In ALG: 2 **+** 3 **×** 4 **=** gives 14. Under an ALG FIX prompt, equals accepts the display setting instead. See sections 2.3 and 4.1.

erf and erfc — error functions

Keys/modes: blue square root = erf; blue e^x = erfc, both modes; **effect U**. Compute erf $X = (2/\sqrt{\pi}) \int_0^X e^{-t^2}\, dt$ and its complement. All finite real arguments are allowed. Use erfc directly for a small complement. Nonreal complex arguments have an approximate magnitude limit of 100 and may fail to converge. Examples from separate reference starts: 0 g $\sqrt{x}$ gives 0; 0 g e^x gives 1. See sections 7.11 and 8.9.

Exponential functions — e to x and 10 to x

Keys/modes: e^x and 10^x, both modes; **effect U**. Raise the named base to X, with complex extensions when enabled. Real inputs have no finite-domain restriction, but exponent range is finite. Angle units do not change these functions; the imaginary argument of e^{a+bi} is in radians. Examples, each from a fresh reference start: 1 e^x displays 2.7183; 3 10^x gives 1000. EEX types an exponent within a number instead. See section 7.3.

Factorial and logarithmic factorial

Keys/modes: $x!$ and f $x!$, both modes; **effect U**. In real mode, factorial requires integer $0 \le X \le 10\,000$, with $0! = 1$. Logarithmic factorial requires a finite nonnegative integer and calculates $\ln \Gamma(X + 1)$ without the 10 000 argument limit. Complex mode uses the Gamma extension, including noninteger real inputs, but retains its poles. Examples from separate reference starts: 5 $x!$ gives 120; 5 f $x!$ displays 4.7875. See sections 7.9 and 7.10.

FIX, FSE, SCI, and ENG

Keys/modes: FIX is orange 7; FSE is orange 8, both modes. RPN FIX takes a whole count from X, commits and drops it, preserving both Last registers. ALG FIX suspends the expression, collects digits, and accepts them with equals. Counts range from zero to working digits. FSE cycles FIX, SCI, ENG while preserving the count and unfinished entry. There are no direct SCI/ENG keys. Formatting leaves numeric values and memories unchanged; FIX can fall back to scientific notation.

Example in RPN: 42 ENTER 2 f 7 displays 42.00; orange 8 then shows 4.20×10^1. ALG selects the same count with f 7 2 = without evaluating its expression. PREC can change the count without using a stack position. See sections 5.3 and 5.4.

FUNC and the argument comma

Keys/modes: FUNC and g FUNC, ALG. FUNC waits for a unary or binary mathematical key, including a shifted one, then inserts its function name and opening parenthesis. It starts an implicit product after an operand. Constants and HYP do not use this prefix. The comma separates two arguments; closing parenthesis ends the call. This does not update ANS until successful equals.

Example from the ALG reference start: FUNC g y^x 2 g FUNC 8) = gives 3, evaluating $\log_2 8$. Backspace cancels a pending FUNC without editing the expression. See section 4.6.

15.5 Alphabetical reference: G through N

Gamma and logarithmic Gamma

Keys/modes: Γ and f Γ, both modes; **effect U**. Compute $\Gamma(X)$ or $\ln\Gamma(X)$. Gamma has poles at zero and negative integers; real log-Gamma is finite where Gamma is finite and positive, not the logarithm of its absolute value. For nonreal inputs, logarithmic Gamma is the principal logarithm of the computed Gamma, not an unwrapped analytic log-Gamma. Examples from separate starts: 5 Γ gives 24; 5 f Γ displays 3.1781. See sections 7.10 and 8.8.

GOAT — play the bundled sound

Keys/modes: g +, both modes. Play the goat audio. There are no numeric operands, results, stack movements, or memory changes, and unfinished entry is preserved. Example: use blue plus with no pending prefix to hear it. You can continue calculating while it sounds. Missing audio or a playback failure reports a message if GOAT is still the latest command; a delayed failure does not replace feedback from later input. This command is independent of scientific or financial calculation. See section 1.5.

HELP — key guide

Keys/modes: g RCL, both modes. Open the Key guide, with Entry, Memory, and Tools topics. Close returns to the same working value or entry; there is no calculation or Last-register change. Example: open HELP, choose Memory, then Close. For precise current prefix routing, use this chapter's key maps and MCLR entry; indexed cash-flow memory uses unshifted CFj after STO/RCL. See section 1.9.

HYP and inverse HYP — hyperbolic function families

Keys/modes: HYP followed by unshifted SIN, COS, or TAN selects sinh, cosh, or tanh; f HYP followed by those keys selects asinh, acosh, or atanh. Both modes; **effect U**. The prefix applies once. Another shift cancels HYP, so do not insert it before the final SIN/COS/TAN key.

Angle settings are irrelevant. Finite real sinh, cosh, tanh, and asinh accept any finite real argument; acosh requires $X \geq 1$, and finite atanh requires $-1 < X < 1$. Complex extensions use the branch conventions in chapter 8. Examples from separate starts: 0 HYP SIN gives 0; 0 HYP COS gives 1; 0 HYP TAN gives 0. 1 f HYP COS gives 0. Use HYP directly in ALG, without FUNC. See section 7.5.

INT and FRAC — integer and fractional parts

Keys/modes: RPN: orange ENTER = INT, blue ENTER = FRAC. ALG: orange $1/x$ = INT, blue $1/x$ = FRAC. ALG equals has no shifted functions. **Effect U**, real-only. INT truncates toward zero; FRAC returns $X - \operatorname{trunc} X$, so a negative input has a negative nonzero fractional part. Display digits have no effect. From separate reference starts, 2.75 +/− followed by orange ENTER gives −2, or blue ENTER gives −0.75. These are unrelated to blue financial i's simple-interest INT. See section 7.6.

INT — simple interest

RPN keys and effect: g i, RPN, real-only. Explicitly store nonnegative days in n, annual percent in i, and signed principal in PV. PMT and FV are unused. Return $(-PV)(i/100)n/360$ in X, $-PV$ in Y, and $(-PV)(i/100)n/365$ in S2. Discard old X and insert these three values above old Y; save old X/Y and preserve all financial inputs. An unlimited stack needs two positions; a fixed depth of at least three retains every result. Fractional days and negative annual rates are accepted if finite.

Example: 90 STO n; 6 STO i; 1000 +/− STO PV; g i gives 15, 1000, and 14.7945 in X, Y, S2. Adding immediately gives principal plus 360-day interest, 1015. See section 13.2.

In ALG, blue i opens Simple interest with Days, Annual interest %, and Present value. The three outputs are labeled; 360-day interest is primary. See section 13.12.

IRR and IRR[a,b] — internal rate of return

RPN keys and effect: IRR for automatic search; f IRR for bounds, RPN. Use the stored cash-flow prefix CF0 through integer n. At least two selected flows, with positive and negative amounts, are required. Automatic search uses stored i as its guess, or zero if i is unset. Bounds are typed lower and upper periodic percentages: finite, strictly ordered, and greater than −100; endpoint NPV must be zero or the interval must bracket a sign change. Bounded search uses that interval instead of the guess.

Lift the periodic percentage into X, preserving both Last registers, all flows, n, and i. IRR does not automatically store its result in i. Multiple roots and roots without a sign change require care; a failed search is not proof that the mathematical problem has no solution.

Example: 100 +/− CF_0; 110 CF_j; 0 STO i; IRR gives 10 percent. Alternatively open orange IRR, set Lower % to 0 and Upper % to 20, and Apply for the same answer. Store the returned rate explicitly before checking it with NPV. See sections 12.7 and 12.8.

In ALG, main IRR opens a worksheet; orange IRR adds bound fields. For direct computation, first finish or clear a pending compound expression, then use CPT IRR to compute from the bank. A lone scalar is allowed. No variant overwrites i with its result. See section 12.9.

LAST X and LAST Y — saved RPN operands

Keys/modes: blue swap = LAST X; blue reciprocal = LAST Y, RPN. Lift the saved operand into X (visible S0) while preserving both Last registers. A Last register holds an earlier operand; it is not the current S0 or S1 position. A new state starts them at zero. Their contents depend on the last command that saved them, not necessarily the last command pressed. They cannot undo bank clears, recover discarded deepest values, or restore a whole earlier stack. With CMPLX off, new Last captures and recalls copy only the real component. Toggling the mode or recalling a Last operand does not change the stored source operand.

Example: 12 ENTER 3 +; f R↓; g $1/x$; g $x \leftrightarrow y$; ÷ gives 4 by recovering the operands of the addition. See section 2.9.

LN, LOG, LOG2, and logarithm to base Y

Keys/modes: orange e^x = LN; orange 10^x = LOG; blue 10^x = LOG2; blue y^x = $\log_Y X$. Both modes. The first three have **effect U**, the last **effect B**. Finite real logarithms require a positive argument; the binary form also requires base $Y > 0$, $Y \neq 1$. Base precedes argument. Complex mode uses principal logarithms, with

$\log_Y X = \operatorname{Log} X / \operatorname{Log} Y$. Logarithm's imaginary component is in radians regardless of angle mode.

Examples from separate starts: 1 f e^x gives 0; 100 f 10^x gives 2; 32 g 10^x gives 5; 2 ENTER 8 g y^x gives 3. See sections 7.3 and 8.8.

L.R. and estimated Y — linear statistics

RPN keys and effect: L.R. fits the stored pairs; f L.R. estimates at X, RPN, real-only. For $\hat{y} = a + bx$, L.R. lifts slope b into X and intercept a into Y, saving old X/Y. It requires at least two observations and positive x variance. Estimate consumes its supplied x, returns $\hat{y}$ in X and correlation r in Y, moves old Y to S2, and saves only Last X. It also requires positive y variance. Both preserve the accumulators. A horizontal line is valid for L.R. but has undefined correlation.

Example: f $\Sigma+$; 2 ENTER 1 $\Sigma+$; 4 ENTER 2 $\Sigma+$; L.R. gives slope 2, intercept 0. Then 3 f L.R. gives estimate 6 and correlation 1. See sections 10.7 and 10.8.

In ALG, STAT provides Linear regression and Linear estimate worksheets. Slope or Estimated Y is primary; companion outputs are labeled. Regression retains a valid horizontal line if correlation is undefined. See section 10.10.

MCLR — select a group to clear

Keys/modes: g STO, both modes. The very next key's group selects the clear before its normal function can run. No individual memory address follows this prefix. The selectors below are convenient representatives of their entire groups.

Table 15.8: MCLR selectors and scope.

Next key	Result
0, either mode	Clear all ordinary and associative memory. Entry/arithmetic keys except backspace select this group.
$\Sigma+$, RPN	Clear statistics. Mean and L.R. do the same.
n, either mode	Clear financial registers and all cash flows. Every Financial-panel key selects this group in either mode.
R↓, RPN	Clear stack and unfinished entry. Swap and reciprocal select the same group.
CANCEL, backspace, f, or g	Cancel without clearing; consume the press.

ALG parentheses, FUNC, CPT, CLEAR, and reciprocal also select ordinary and associative clearing. Scientific keys, STO/RCL, and ALG ANS cancel MCLR

without acting. Memory-bank clears preserve the exact unfinished entry, stack, Last registers, and settings; stack clear preserves both Last registers and banks. Example: g STO 0 removes all ordinary and associative entries, not only R0. Use CANCEL, backspace, either shift, or physical Escape to cancel without clearing. See section 6.10 and chapter 14.

MDY and DMY — date order

Keys/modes: orange CFj = MDY; blue CFj = DMY, both modes. Select month-first (Month.DDYYYY) or day-first (Day.MMYYYY) interpretation. Stored numbers, unfinished entry, stack, and Last registers do not change. This shared saved setting governs date and bond calculations. Example: orange CFj selects MDY, under which 2.032026 means February 3, 2026; blue CFj would interpret the same number as March 2. It does not convert one encoding into the other. See section 13.3.

Mean and weighted mean

RPN keys and effect: $\bar{x}$ and orange $\bar{x}$, RPN, real-only. Mean lifts $\sum x/n$ into X and $\sum y/n$ into Y, saving old X/Y. Weighted mean returns $\sum xy/\sum y$ by replacing X, saving only Last X. Both require data; weighted mean also requires nonzero total weight. Y-coordinates are weights only for weighted mean. The bank is unchanged.

Example: f Σ+; 2 ENTER 10 Σ+; 3 ENTER 20 Σ+. Unshifted mean returns X = 15, Y = 2.5; orange mean then gives 16, since $(2\times10+3\times20)/(2+3) = 16$. Negative weights are accepted; SD and regression still count each pair once. See section 10.5.

In ALG, select Mean or Weighted mean under STAT. Mean X is the primary paired output. See section 10.10.

MOD and REM — two remainder rules

Keys/modes: blue division = MOD, blue multiplication = REM, both modes; **effect B**, real-only. With dividend Y and nonzero divisor X, return $Y - qX$. MOD truncates Y/X toward zero; REM chooses the nearest integer quotient, ties even. MOD's nonzero remainder has Y's sign; REM may have the opposite sign. Examples from separate starts: 7 ENTER 4 followed by blue division gives 3, or blue multiplication gives −1. Neither function is restricted to integers. In a worksheet field expression, a typed percent symbol denotes truncating remainder; the calculator's percentage key is a different binary function. See section 7.7.

n, i, PMT, PV, FV — financial register keys

RPN keys and effect: five main Financial keys, RPN. A bare key stores X after number entry, ordinary arithmetic, stack actions, or recall. After a financial-register store, a TVM solve, or the 12-times/divide shortcuts, the next bare key solves its variable. BEGIN/END preserves this readiness. Explicit STO always stores, and RCL always recalls. There is no store/solve readiness annunciator.

Storage commits entry and preserves stack and both Last registers. A solve uses all four other registers, treating unwritten ones as zero; it replaces the target register and lifts the answer into X while preserving both Last registers. n counts periods, i is percent per period, and PV, PMT, FV use opposite signs for money paid and received. All participating values must be finite; $i > -100$, and n must be nonnegative, positive when solving i or PMT. Old target contents are ignored, including old i as a guess. Solved n is rounded to a whole period: a fractional part below 0.005 normally rounds down, otherwise up; a small numerical guard treats values extremely close to the cutoff as boundary cases. See section C.6 for the precise rule. Entered fractional n retains the continuous compound-annuity convention.

Example: select END and follow the sequence without inserting ENTER before the final FV solve: f FV; 2 n; 10 i; 100 +/− PV; 0 PMT; FV gives 121 and stores it in FV. For unambiguous solving after inspecting inputs, recall and explicitly restore a known input, then press the target. See chapter 11.

In ALG, each bare register key always stores the displayed operand; RCL recalls and CPT explicitly solves. A store preserves pending arithmetic. See section 11.12.

NINT — nearest integer

Keys/modes: g 9, both modes; **effect U**, real-only. Round to the nearest integer with an exact tie to even, independently of display digits. Examples, each from a fresh reference start: 2.5 g 9 gives 2; 3.5 g 9 gives 4. RND instead rounds to decimal places with ties away from zero. See sections 5.6 and 7.6.

NPV — net present value

RPN keys and effect: NPV, RPN, real-only. Require a nonempty flow list, an explicitly stored final index n, and explicitly stored periodic percentage $i > -100$. Calculate $\sum_{t=0}^{n} \mathrm{CF}_t/(1 + i/100)^t$. Lift the result into X, preserving both Last registers, all financial registers, and cash flows. It does not store the result in PV. n = 0 uses only CF0; BGN does not shift the explicit times.

Example: 100 +/− CF_0; 110 CF_j; 10 STO i; NPV gives zero to ordinary monetary precision, possibly with a tiny nonzero floating-point residue. See sections 12.5 and 12.6.

In ALG, NPV opens its input worksheet. For direct computation, first finish or clear a pending compound expression; a lone scalar is also allowed. Then use CPT NPV to calculate from n, i, and the selected cash-flow prefix. See section 12.9.

15.6 Alphabetical reference: P through S

Parentheses — explicit ALG groups

Keys/modes: (and), ALG. Delimit a group; an opening group after an operand inserts multiplication. Closing a group previews its value but does not update ANS or evaluate the entire surrounding expression. Equals does not supply missing closing parentheses. Example from the ALG reference start: (2 + 3) × 4 = gives 20. Groups are limited to 128 levels. See section 4.2.

Percent, percent change, and percent of total

Keys/modes: blue 1 = %, blue 2 = Δ%, blue 3 = T%, both modes; **effect P**. Return respectively $YX/100$, $100(X-Y)/Y$, and $100X/Y$. The last two need a nonzero base Y. Rates and answers use percent units: 10 means 10 percent. Complex mode extends the same formulas. In ALG each is an ordinary binary operation with no retained base.

Examples from separate starts: 200 ENTER 10 g 1 gives X = 20, retaining Y = 200; 80 ENTER 92 g 2 gives 15; 200 ENTER 50 g 3 gives 25. See section 7.8.

Permutations and combinations — nPr and nCr

Keys/modes: blue Gamma = nPr; blue factorial = nCr, both modes; **effect B**, real-only. Put the number available n in Y, number selected r in X. Require integers $0 \le r \le n \le 10\,000$. Return $n!/(n-r)!$ for ordered selections or $n!/[r!(n-r)!]$ for unordered selections without replacement. Both give 1 when r = 0. From separate starts, 10 ENTER 3 followed by blue Gamma gives 720, or blue factorial gives 120. Large integer results can still exceed working precision. See section 7.9.

Power and indexed root

Keys/modes: y^x and its orange $\sqrt[x]{y}$, both modes; **effect B**. Power uses base Y, exponent X. In real mode, a negative base needs an integer exponent for a real result. Root uses radicand Y, nonzero index X; a negative radicand requires an odd integer index. Integer indices are limited to $\pm 1\,000\,000$. A negative index takes the reciprocal root. A noninteger real index uses $Y^{1/X}$ for $Y \geq 0$. Zero to a negative power has no finite result. Complex mode returns principal powers/roots and permits complex indices except zero.

Examples from separate starts: 2 ENTER 5 y^x gives 32; 125 +/− ENTER 3 f y^x gives −5 in real mode. ALG powers associate from the right: $2^{3^2} = 512$. See sections 4.3, 7.2 and 8.8.

PRC and YTM — bond price and yield

RPN keys and effect: orange PV = PRC; blue PV = YTM, RPN, real-only. First select the date order and explicitly store nonnegative annual coupon percent in PMT. For PRC, also store annual yield percent $i > -200$; for YTM, store positive clean price per 100 in PV. Then enter settlement in Y and later maturity in X. n and FV are ignored. Coupons are semiannual, redemption 100, and the schedule is anchored to maturity using actual coupon-period fractions. One remaining coupon uses simple yield for the partial period; longer schedules use compound discounting.

PRC replaces the dates with clean price X and accrued interest Y, and stores clean price in PV. YTM consumes both dates, returns annual yield in X, and stores it in i. Both save original X/Y. No old i guess or user bracket is used by YTM. Example, MDY: store 6 STO PMT and 6 STO i; clear the stack and enter 1.012024 ENTER 1.012026 f PV. PRC returns clean 100, accrued 0. Clear and re-enter those dates with g PV; YTM returns 6, using the clean price just stored. See sections 13.6 and 13.7.

In ALG, the same keys open labeled bond worksheets. PRC returns Price per 100 and Accrued interest and stores PV; YTM returns Annual yield % and stores i. See section 13.12.

PREC — working and display precision

Keys/modes: g 7, both modes. Enter whole Working digits from 16 through 10 000 and Display digits from zero through the proposed working count, then Apply. Change subsequent arithmetic precision and presentation count without committing entry, evaluating an expression, moving the stack, or rewriting existing numbers. Retain the chosen FIX/SCI/ENG notation. Invalid fields leave settings unchanged and the panel open.

Example: open PREC, set working 80 and display 8, and Apply; a subsequent `1.25` **ENTER** under FIX shows `1.25000000`. Raising precision cannot restore lost digits of an old value. The help label INFIX refers to working digits, not ALG. See sections 5.2 and 5.7.

RAN# — random number

Keys/modes: g 4, both modes; **effect Z**. Generate a real value $0 \le u < 1$ from random decimal digits at working precision. No seed or repeat-sequence control is exposed. Example: blue 4 from the reference start puts one draw in X; another blue 4 puts a new draw in X and the first in Y. Repetition is possible. FIX 4 can display 1.0000 after rounding a value below 1. ALG insertion follows the constant rule. See section 7.12.

RCL and STO — ordinary and financial memory

Keys/modes: RCL or STO, then one digit 0–9, both modes. Both also accept unshifted n, i, PMT, PV, FV as financial targets, and CFj with its separate indexed rules. In ALG, CFj opens a worksheet rather than consuming stack operands. Select the target to commit an active number. STO copies the current operand and preserves stack and Last registers; in ALG it stores the current operand/preview without evaluating the whole expression.

Ordinary RPN RCL lifts the value and saves old X in Last X, preserving Last Y. Financial RCL also lifts but preserves both Last registers. ALG RCL inserts or replaces the trailing operand; it does not update ANS. An unset register recalls zero without becoming occupied. Stored zero remains occupied. STO followed by $+, -, \times, \div$ and a target changes that destination from M to M op v, leaving the current operand v in place; division by zero is rejected. Stores and recalls copy full complex pairs with CMPLX on; with it off, they write purely real values. Recall leaves the source memory unchanged. Arithmetic storage also follows the current complex setting; see section 6.5.

Example: 20 **STO** 1; 5 **STO** − 1; **RCL** 1 returns 15. In RPN, financial storage arms the next bare TVM key to solve; recall arms it to store. In ALG, bare financial keys always store and CPT explicitly computes. See chapter 6.

RCLΣ — copy statistical accumulators

RPN keys and effect: g L.R., RPN. Commit entry, then copy the six accumulators into ordinary R4–R9 in this order:

$$n, \quad \sum x, \quad \sum x^2, \quad \sum y, \quad \sum y^2, \quad \sum xy.$$

Overwrite those six ordinary registers, without moving the stack or changing either Last register or the statistical bank. Later data changes do not refresh the copies, and editing them does not edit statistics. There is no inverse load command.

Example: f Σ+; 2 ENTER 1 Σ+; g L.R.; RCL 9 gives 2, the sum of products. Copying an empty bank writes six zeros and lights MEM without lighting STAT. See section 10.9.

In ALG, choose Recall summations under STAT, then RECALL SUMMATIONS. The same copy to R4–R9 occurs, all six totals are labeled, and count is primary. See section 10.10.

Reciprocal — one divided by X

Keys/modes: $1/x$, both modes; **effect U**. Return $1/X$, including the full complex reciprocal when enabled. A zero denominator has no finite answer. Example: 8 $1/x$ gives 0.125. In ALG its orange and blue functions are INT and FRAC; in RPN those layers are CLEAR and LAST Y. See section 7.2.

RND — round the working value to decimal places

RPN keys and effect: g PMT, RPN, real-only. Round X to the current display count of decimal places, with ties away from zero. Replace X and save its old value in Last X; preserve deeper values, Last Y, and source memories. The count is decimal places in the value even when SCI or ENG is selected. This changes a value, unlike formatting. Example: 2 f 7; .125 g PMT gives 0.13, whereas FIX 2 alone displays exact 0.125 as 0.12 by ties-to-even rounding. Store the result explicitly to replace a financial payment. See sections 5.6 and 11.13.

In ALG, first finish or clear a pending compound expression; a lone scalar is also allowed. Then use the same key to round the displayed operand, display a completed result, and update ANS. The mathematical rounding rule is unchanged.

Roll down and roll up

Keys/modes: R↓ and its blue R↑, RPN with a fixed stack. Commit entry and cyclically move every value. Down sends Y to X and old X to the deepest position; up sends the deepest value to X. No values or Last registers are replaced. Roll is unavailable in unlimited mode. Example: 1 ENTER 2 ENTER R↓ shows 1; blue roll then restores 2 in X. See section 2.7.

RPN and ALG — workspace selection

Keys/modes: header mode buttons. Select the independent stack or expression workspace without evaluating, committing, or copying an operand. Shared memories and settings remain shared. Mode selection normally clears shifts and pending commands; an ALG FIX prompt consumes the first press as cancellation without switching. Example: type `12.30` in RPN, select ALG, then return to RPN; the unfinished literal still reads 12.30. Store and recall ordinary memory to transfer a value deliberately. See section 1.7.

SD — sample standard deviations

RPN keys and effect: g $\bar{x}$, RPN, real-only. Use at least two stored observations; lift x sample deviation into X and y sample deviation into Y, saving old X/Y. Preserve all accumulators. Each variance divides the centered sum of squares by $n-1$. Constant coordinates return zero; negative implied variance is rejected. This is neither population nor frequency-weighted SD.

Example: f $\Sigma+$; 0 ENTER 8 $\Sigma+$; 0 ENTER 10 $\Sigma+$; 0 ENTER 12 $\Sigma+$; g $\bar{x}$ gives X = 2, Y = 0. See section 10.6.

In ALG, choose Sample standard deviation under STAT. The X deviation is primary and Y is separately labeled; no stack operands are used. See section 10.10.

Shifts and prefix cancellation

Keys/modes: f and g, both modes. Select a one-key alternate layer; repeating the same shift cancels, and pressing the other selects that layer. Shifts preserve STO, RCL, and FUNC while allowing a shifted target; they cancel HYP. Missing alternate legends report an error without performing the main action. Example: blue 7 opens PREC; Close returns without applying a setting.

Header CANCEL dismisses the entire pending selection without editing, including simultaneous shifts and prefixes. Unshifted backspace, CLx, or CLEAR also cancels an ordinary pending prefix instead of editing. For HYP, use CANCEL or unshifted backspace because a shift first cancels HYP and arms its own layer. MCLR consumes backspace or either shift as cancellation without clearing anything. Physical Escape cancels any current shift, prefix, or error before performing CLx on a later press. See CANCEL (p. 194), section 1.6 and chapter 14.

SHOW — inspect and recall cash flows

RPN keys and effect: g IRR, RPN. Open Cash flows; use the index stepper to inspect the entire stored list, including any entries after the NPV/IRR cutoff n. The selected amount uses up to 32 significant digits. Inspection and Close preserve entry, stack, n, and Last registers. Recall into X commits entry and replaces X without lift or Last-register changes. An empty bank cannot be recalled.

Example: 100 +/− CF_0; 110 CF_j; open SHOW, select index 0, and choose Recall into X to obtain −100, with n still 1. This panel does not edit amounts or frequencies. See section 12.4.

In ALG, SHOW opens the indexed Recall cash flow worksheet. Accept an index and choose RECALL; Use value inserts the selected amount. Opening alone does not read or insert a value. See section 12.9.

Sigma plus and minus — add or remove a pair

RPN keys and effect: Σ+ and its blue Σ−, RPN, real-only. For single-variable data use y = 0 for each observation. Supply x in X and y in Y by entering y first, ENTER, then x. Add or subtract the pair's contributions to the six accumulators; replace X with the new count, preserve Y and deeper values, and save both original operands in Last X/Y. Removal requires a positive count but cannot verify that the pair was previously entered.

Both sample real components and all six existing and proposed totals must be finite. A rejected addition or removal shows an error and preserves the bank, committed stack values, and Last X/Y; active entry is finished before validation. Imaginary components remain ignored.

Example: f Σ+; 4 ENTER 2 Σ+ gives count 1. Re-enter 4 ENTER 2 g Σ+ to return the count to zero. STAT remains lit until the bank is cleared. The bank stores sums, not a retrievable sample list; use finite values. See sections 10.2 and 10.4.

In ALG, choose Add sample or Remove sample under STAT. Accept X and Y, then ADD SAMPLE or REMOVE SAMPLE. Validation precedes all accumulator updates. See section 10.10.

Square and square root

Keys/modes: orange square root = x^2; unshifted $\sqrt{x}$, both modes; **effect U**. Square multiplies X by itself. Real square root requires $X \geq 0$ and returns the nonnegative root. Complex square root uses the principal branch, with $\sqrt{-r} = +i\sqrt{r}$ on the negative real axis. Examples from separate starts: 12 f $\sqrt{x}$ gives 144; 81

$\sqrt{x}$ gives 9. Squaring then taking a real square root returns absolute value, not the original sign. See sections 7.2 and 8.8.

STACK and full-stack inspection

Keys/modes: f $x \leftrightarrow y$ opens RPN stack depth; the display triangle opens the RPN inspector. STACK accepts a typed depth of 2–1000, or 0 for unlimited. Apply commits active entry; shrinking retains the positions nearest X and discards deeper values, while enlarging adds zero positions. Last registers and memories remain unchanged. Example: open STACK, enter 4, Apply to select a fixed four-register stack. No number is entered on the stack by typing in that field.

The inspector is read-only, preserving entries and pending commands. Deeper positions appear above X/S0; each row can scroll horizontally. Close with its collapse control, outside tap, or physical Escape. Example: type 12.30 and open the triangle to inspect the unfinished X without committing it. See sections 2.6 and 2.10.

SWAP — exchange X and Y

Keys/modes: $x \leftrightarrow y$, RPN. Commit entry and exchange the two nearest complete operands, preserving deeper positions and both Last registers. Unlimited mode requires two positions. This is separate from orange 6's complex-component swap. Example: 3 ENTER 12 $x \leftrightarrow y$ ÷ gives 4, calculating 12/3. See section 2.7.

15.7 Alphabetical reference: T through Z and numeric legends

Trigonometric functions and their inverses

Keys/modes: SIN, COS, TAN and their blue inverse functions, both modes; **effect U**. The current DEG/RAD/GRAD setting controls argument and inverse-result units. In ALG the function retains the unit selected at insertion. In complex mode both components of a trig argument or inverse result are scaled to that unit; these are not hyperbolic functions.

Use equivalent ranges in RAD or GRAD. Examples, each from a fresh reference start with DEG selected: 30 SIN gives 0.5; 60 COS gives 0.5; 45 TAN gives 1; .5 g SIN gives 30; .5 g COS gives 60; 1 g TAN gives 45. The inverse legend means inverse function, not reciprocal. Complex branches and numerical precision conventions are in sections 8.8 and 8.9; real use is in section 7.4.

Table 15.9: Finite real trig domains and inverse ranges.

Function	Domain or returned range
SIN, COS	Any finite real angle.
TAN	Exclude odd right-angle poles; nearby values are sensitive.
Inverse SIN	Input $[-1, 1]$; result $[-90, 90]$ degrees.
Inverse COS	Input $[-1, 1]$; result $[0, 180]$ degrees.
Inverse TAN	Any finite real input; result strictly between -90 and 90 degrees.

12 times and 12 divide — monthly TVM shortcuts

RPN keys and effect: orange n multiplies X by 12 and stores n; orange i divides X by 12 and stores i. RPN, real-only. Each replaces X without movement or Last-register changes, then arms the next bare TVM key to solve. Require a finite result. They do not set a calendar or payments-per-year option. Example: 30 f n sets n and X to 360; 6 f i sets i and X to 0.5. Division by 12 converts a nominal annual monthly rate, not an effective annual rate. See section 11.2.

In ALG, first finish or clear a pending compound expression; a lone scalar is also allowed. Then use the same shortcuts to store the scaled operand in n or i and publish a completed result with ANS updated. See section 11.12.

Finding an alternate legend by name

Several alphabetized families above cover more than one legend. table 15.10 provides a direct route without implying additional commands.

Table 15.10: Alternate names and symbols in this reference.

Legend or name	Entry
ΔDAYS	DATE and day differences, 198.
Δ%, T%, %	Percent family, 207.
Γ, $\ln\Gamma$	Gamma family, 201.
Σ+, Σ−	Sigma family, 212.
$\bar{x}$, $\overline{xw}$	Mean family, 205.
$\hat{y}$, correlation	L.R. and estimated Y, 204.
π, Euler's e, $\ln 2$	Constants, 197.
CF0, CFj, Nj	Cash-flow entry, 194.
CPT, STAT (ALG)	Explicit computation and worksheets, 197.
DB, DB→SL, SL, SYD	Depreciation, 198.
DMY, MDY	Date order, 205.
END	BGN toggle, 194.
ENG, SCI, FSE	Formatting, 200.

Legend or name	Entry
FRAC	INT and FRAC, 202.
FV, i, n, PMT, PV	Financial registers, 206.
NINT, floor, ceiling	Integer rounding, 206 and 195.
P↔R, Re/Im, Re↔Im	Complex controls, 196.
REM	MOD and REM, 205.
STO_{xy}, RCL_{x}	Associative memory, 193; ALG worksheets, 64.
YTM	PRC and YTM, 208.
$x!$, $\ln x!$	Factorial family, 200.
x^2, $\sqrt{x}$	Square and square root, 212.
y^x, $\sqrt[x]{y}$	Power and indexed root, 208.
$\log_y x$, LOG2	Logarithms, 203.

15.8 Display flags and pending indicators

The main display and expanded stack use the same indicators. A memory flag reports occupied storage, not validity, a count of entries, or which operation will run next. Explicitly stored zero counts as occupied; recalling an unset register does not. The same bank flags appear in both modes because the banks are shared.

Table 15.11: Current flags and their meanings.

Indicator	Meaning and how it ends
FIX, SCI, ENG + count	Selected notation and fractional-place count. Always shown; FIX may use scientific fallback without changing its flag.
DEG, RAD, GRAD	Current angle unit. Always shown.
MDY, DMY	Current packed date order. Always shown.
COMPLEX	Complex calculation and two-component display enabled; orange division toggles it.
BGN	BEGIN payments. Absent means END; blue n toggles timing.
f, g	Active next-key shift, highlighted in its color. Same shift again cancels; selecting an ordinary key normally releases it.
ENTRY	Active RPN numeric/component entry, or the whole unfinished ALG expression, including between numeric entries. Successful ALG equals ends it; RPN commits end numeric entry.
STO	Waiting for a storage target. Complete the target or cancel the prefix.

Indicator	Meaning and how it ends
STO+, STO−, STO×, STO÷	Arithmetic storage modifier selected; next valid target receives old memory op current operand.
RCL	Waiting for a recall target.
HYP, INVHYP	Waiting for unshifted SIN, COS, or TAN; one selection only.
FUNC	ALG waiting for a unary or binary mathematical function key.
CPT	ALG waiting for n, i, PMT, PV, FV, NPV, or IRR to compute.
ΔFIX	Separate ALG display-count prompt. Digits then equals accept; another enabled key cancels without its normal action.
MCLR	Next key selects a group clear. CANCEL, backspace, either shift, or Escape cancels without clearing.
MEM	At least one ordinary register is stored, including zero.
ASSOC	At least one associative key/value entry is stored.
STAT	Statistical accumulators occupy the bank, even if count is zero.
FIN	At least one financial register is stored, including n = 0.
CF	The cash-flow list contains at least one entry, including zero.

A restored state can also contain **TURN**, an angle unit not offered by the current DEG/RAD/GRAD keys, or **DIGITS**, an older notation count prompt. These are not extra current keyboard selections. DRG returns TURN to DEG; DIGITS accepts one unshifted digit 0–9 or can be canceled as an ordinary prefix. A selected turn in the conversion menu is a unit choice and does not select the TURN angle setting.

There is no dedicated indicator for TVM store/solve readiness, a particular occupied register, a depreciation crossover year, or an HP odd-period C setting. Read the status line as well as the flags. CF0 and CLcf store n = 0 and therefore light FIN; clearing only cash flows does not extinguish it. See section 6.1.

15.9 Settings, defaults, and tool fields

The following defaults describe a new calculator state. Relaunch restores saved work; it is not a reset to this table. RPN and ALG retain independent operands and editors, but precision, display, complex settings, angle unit, date order, payment timing, and memory banks are shared.

Table 15.12: Settings and new-state defaults.

Setting	Default	Control and scope
Entry mode	RPN	Header buttons; independent working areas.
Panel	Scientific	Swipe top two portrait rows; landscape shows both.
RPN stack	Fixed 5, zeros	STACK; 0 unlimited or fixed 2–1000. Last X/Y also start at zero.
ALG workspace	Empty; ANS = 0	Equals, worksheet ENTER, and computation update ANS; CLEAR preserves it.
Working precision	600 decimal digits	PREC, 16–10 000. Existing values retain creation precision.
Notation/count	FIX 4	FIX, FSE, PREC; count means fractional places.
Angle unit	DEG	Direct DEG/RAD/GRAD or DRG cycle.
Complex mode/form	Off; rectangular	Orange division and orange 5. Mode/form changes commit numeric entries in both workspaces.
Date order	MDY	Orange/blue CFj in either mode.
Payment timing	END	Blue n toggles BGN; bank clears preserve timing.
Memory banks	Unoccupied	Store or domain commands create entries; MCLR removes selected banks.
Digit ruler	Enabled when needed	Header CAL toggles its visibility when a number overflows.
Conversion choices	Length; meter to foot	CONV remembers menu choices. Changing dimension selects that family's first and second units.
IRR bounds fields	RPN: 0 and 2 percent; ALG: 0 and 100 percent	Orange IRR opens a panel in RPN or a worksheet in ALG. Each retains its own bounds; inspect them before use.
Cash-flow view	Index 0	RPN SHOW remembers a stepper selection, clamped to the current list when opened. ALG SHOW retains a worksheet index; RECALL rejects an index outside the list.

Entry punctuation	Decimal point, comma grouping	Calculator entry uses the point; there is no keyboard setting for locale punctuation.

ALG worksheets use separate accepted inputs and expression drafts. Equals evaluates a field, ENTER accepts it, and the named action computes from accepted inputs. Use value transfers a selected output; CANCEL discards unaccepted edits. See section 4.12.

The precision and depth panels refresh their fields from the actual current settings when opened. Editing a field does not place a value on the calculator stack. PREC Apply preserves unfinished entry; STACK Apply commits it before changing depth. Conversion Apply commits and replaces the active operand. In RPN, bounded IRR Apply calculates and lifts a result. RPN SHOW's Recall into X commits and replaces X; its stepper only inspects. In ALG, bounded IRR and SHOW follow the worksheet ENTER, COMPUTE or RECALL, and Use value procedure. A tool's Close control does not perform its operation. Invalid settings keep the panel available for correction without partially changing precision settings.

Numeric entry and all state-changing actions are saved locally as you work, including unfinished literals, prefixes, and panel inputs. There is no Save button, user undo/history browser, cloud synchronization, or single factory-reset key in this interface. Grouped clearing retains settings. For persistence failures and recovery, see chapter 14; for display rounding and old-value precision, see chapter 5.

15.10 Complete conversion-unit inventory

Open CONV with blue 8, select Dimension first, then From and To. The names below identify every selectable unit. Each conversion stays within one family; no cross-family inference or stored unit tag is provided. Chapter 9 explains exact and approximate factors, US/Imperial distinctions, offsets, and examples; chapter C gives the numerical definitions.

Table 15.13: All fifteen conversion families and their units.

Family	Unit names and symbols
Length	meter (m), kilometer (km), centimeter (cm), millimeter (mm), micrometer (μm), nanometer (nm), inch (in), foot (ft), yard (yd), mile (mi), nautical mile (nmi).
Area	square meter (m^2), square kilometer (km^2), square centimeter (cm^2), square millimeter (mm^2), square inch (in^2), square foot (ft^2), square yard (yd^2), acre (acre), hectare (ha).

Family	Unit names and symbols
Volume	liter (L), milliliter (mL), cubic meter (m^3), cubic centimeter (cm^3), cubic inch (in^3), cubic foot (ft^3); US fluid ounce (US fl oz), US pint (US pt), US quart (US qt), US gallon (US gal); Imperial fluid ounce (Imp fl oz), Imperial pint (Imp pt), Imperial quart (Imp qt), Imperial gallon (Imp gal).
Mass	kilogram (kg), gram (g), milligram (mg), metric tonne (t), ounce (oz), pound (lb), stone (st), US short ton (short ton), Imperial long ton (long ton).
Temperature	kelvin (K), degree Celsius (°C), degree Fahrenheit (°F), degree Rankine (°R).
Speed	meter per second (m/s), kilometer per hour (km/h), mile per hour (mph), foot per second (ft/s), knot (kn).
Acceleration	meter per second squared (m/s^2), foot per second squared (ft/s^2), standard gravity (g_0).
Force	newton (N), kilonewton (kN), pound-force (lbf).
Pressure	pascal (Pa), kilopascal (kPa), megapascal (MPa), bar (bar), millibar (mbar), standard atmosphere (atm), pound per square inch (psi), torr (Torr).
Energy	joule (J), kilojoule (kJ), megajoule (MJ), watt-hour (Wh), kilowatt-hour (kWh), thermochemical calorie (cal), thermochemical kilocalorie (kcal), International Table BTU (BTU).
Power	watt (W), kilowatt (kW), megawatt (MW), mechanical horsepower (hp).
Time	second (s), millisecond (ms), microsecond (μs), nanosecond (ns), minute (min), hour (h), day (d).
Frequency	hertz (Hz), kilohertz (kHz), megahertz (MHz), gigahertz (GHz).
Angle	radian (rad), degree (°), gradian (grad), turn (turn).
Torque	newton-meter ($N \cdot m$), pound-foot ($lbf \cdot ft$).

Temperature choices convert readings with offsets, not temperature intervals; readings below absolute zero are not screened for physical plausibility. Mass ounces are avoirdupois, and US volume choices are liquid measures. A day in the Time family is a fixed 24 hours, unrelated to date arithmetic or civil time zones. Angle conversion is independent of the trig angle setting and does not change it. Conversions use real components even in complex mode, including a same-unit conversion.

15.11 Domains and operational limits

Use the individual entries for their mathematical domains. This table collects capacities and restrictions that cannot be changed by selecting more display digits. Working precision is finite and does not guarantee the same number of correct answer digits, particularly for sensitive formulas and iterative results.

Table 15.14: Exposed limits and important qualifications.

Area	Bound or convention
Precision	16–10 000 working decimal digits; display count 0–working digits. 600 digits corresponds to 1994 MPFR bits.
Stack	UI fixed depth 2–1000, or unlimited. Unlimited has no fixed discard boundary but is still constrained by device resources. Roll requires a fixed stack.
Ordinary memory	Ten addresses, R0–R9. STO/RCL take one target digit.
Associative memory	Real-component keys in both modes; a NaN real component is forbidden. Equal real keys identify one entry. No user-selectable capacity or individual delete command.
ALG expression	4096 tokens; 128 nested groups; streaming numeric literal 10 000 characters. A worksheet field expression is limited to 100 000 characters before parsing and the same token/group limits.
ALG FIX prompt	Counts within working digits; input collection is bounded to 128 digit characters, including leading zeros.
Real-mode factorial	Integer argument 0–10 000. Log-factorial instead accepts finite nonnegative integers without that argument work bound.
Counting functions	Integers $0 \le r \le n \le 10\,000$. Real components are used even in complex mode.
Real indexed root	Nonzero integer index within $[-1\,000\,000, 1\,000\,000]$, or the noninteger nonnegative-radicand power route described in its entry.
Complex special functions	Nonreal erf/erfc has an approximate magnitude limit of 100 and can fail to converge. Complex Gamma supports the exposed precision range. These custom algorithms use MPC arithmetic but retain their approximation and convergence limits.

Area	Bound or convention
Statistics	Six raw accumulators, not saved observations. At least one pair for means, two for SD/regression/estimate; variance and weight restrictions apply. Addition/removal rejects nonfinite real sample components or existing/proposed totals without changing the bank.
TVM	Rate greater than −100 percent per period; n nonnegative, positive for i/PMT solves. No HP initial odd-period C setting. Unwritten participating TVM registers supply zero.
Cash flows	1000 expanded entries including CF0, indices 0–999; Nj 1–1000 subject to remaining capacity. NPV/IRR select the prefix through explicitly stored integer n.
AMORT	Additional payment count 0–100 000; stored n must be a nonnegative integer. No automatic payoff stop or per-payment rounding.
Dates	Gregorian years 1–9999. Packed encoding has six fractional digits. DATE day offset within ±3 652 058, with the resulting date also within range.
Bonds	Semiannual coupons; redemption 100; actual coupon fractions; maturity after settlement. PRC yield greater than −200 percent; YTM requires positive clean price.
Depreciation	Positive life, integer year at least 1, cost at least nonnegative salvage. Positive DB factor; active DB calculation at most 100 000 annual steps. Final partial-year conventions apply.
Financial iterative solves	Standard tolerance 10^{-30}, 200 iterations per search stage, 100 000 evaluation work budget. Tolerance applies to the numerical method, not a promise of 30 or 600 correct digits in a sensitive financial answer.

Numerical magnitude is not limited to $10^{\pm p}$ when working precision is p digits. Exponent range is a separate, finite property; increasing p does not generally cure overflow or underflow. Ordinary real operations and each MPC primitive's components use nearest, ties-even rounding. Display rounding is also nearest, ties even; financial RND uses ties away from zero, and solved TVM n has its separate 0.005 rule. Complex and financial formulas composed from multiple operations do not inherit a blanket guarantee of correct rounding for the entire calculation.

In complex mode, integer/fraction operations, remainders, counting, atan2, conversions, statistics, and finance use real components. ABS instead uses the complete complex magnitude. With complex mode off, scientific calculations use real com-

ponents while stored pairs can remain intact. Branch choices can make a complex root differ from the real odd root. See chapters C, 5 and 8.

15.12 Errors, warnings, and recovery cues

Messages identify different kinds of failure. A domain error does not mean a stored bank was cleared, and a numerical search failure does not prove that an equation lacks a root. An active entry can be committed before a command fails; check ENTRY before expecting backspace to remove one digit. The following messages are grouped by the correction they require rather than by internal error names.

Table 15.15: Messages and the next useful check.

Message or message family	Meaning and correction
This key has no orange/blue function	That layer is unassigned; the main command did not run. Check the map and retry the intended layer.
Finish the expression first	Complete or clear the pending compound ALG expression before a direct financial computation. The expression is preserved.
Press ENTER to accept … first	Worksheet computation uses accepted inputs. ENTER each edited field before trying the action again. CANCEL closes the worksheet and discards its unaccepted edits; reopen it to compute from retained accepted inputs.
Choose 0–9, n, i, PMT, PV, FV, or CFj	STO/RCL needs one of these unshifted targets. ALG CFj opens an indexed worksheet.
Choose a unary or binary math function	FUNC expects a mathematical operation, not a constant, HYP prefix, or settings key.
HYP requires SIN, COS, or TAN	The pending hyperbolic selection was followed by an unsuitable key. Re-enter the prefix and intended key.
FIX requires an integer…; FIX canceled	Check the count against working digits. ALG requires FIX, digits, equals; other enabled keys cancel its separate prompt.
Use 16–10000 working digits; Use 0–… display digits	Correct both PREC fields before Apply; display cannot exceed proposed working precision.
Use 0 for unlimited, or a fixed depth from 2 through 1000	Correct the typed STACK depth; one is not an exposed fixed-depth choice.
RPN operation requires … operands	The unlimited stack has too few positions. Supply the required operands rather than increasing precision.

Message or message family	Meaning and correction
Rotate is undefined for an infinite RPN stack	Select a fixed depth if cyclic roll is needed, or inspect the unlimited stack directly.
Finish the numeric mantissa and exponent before evaluation	An ALG number is incomplete. Supply exponent digits or correct the entry.
An operand/operator is missing; Parentheses are unbalanced	Inspect the ALG expression and supply or correct the missing element. Equals does not close groups automatically.
Wrong number or placement of function arguments	Use a comma only inside a binary function call and supply exactly the required arguments.
Expression exceeds…; Expression nesting exceeds…	Split the calculation into smaller parts and retain useful results in ordinary memory.
The expression has no finite result	ALG encountered NaN or infinity. Check domains and divisors; ANS retains the last successful result.
Enter an integer from…; Log factorial requires a nonnegative integer	A counting, root-index, or integer-only input is outside its stated range. Check the actual operand, not just its rounded display.
A zeroth root is undefined; No real root…	Check root index and radicand, and whether the intended answer is real or principal complex.
Logarithm requires X greater than zero and a positive base Y other than 1	Check argument and base order in real mode.
The base percentage value cannot be zero	Percent change and share need a nonzero original amount or total in Y.
Used real portion only	A real-only operation discarded nonzero imaginary data while complex mode was active. The answer is a real result; storage alone may still retain the original pair.
Enable CMPLX…; nonfinite component editing error	Enable complex mode before component controls. Replace or clear a NaN/infinite value before reopening it for component entry.
Complex erf/erfc requires magnitude at most 100; convergence message	The nonreal path has an approximate magnitude boundary and a work restriction. Real arguments have a different path; more displayed places do not remove the restriction.
Statistics samples must be finite	Replace NaN or infinity in the real component of X or Y. The pair was not added or removed.
Statistics totals must be finite	An existing total is nonfinite, or updating the sums would produce a nonfinite total. The bank is unchanged; clear and re-enter data if its existing totals are nonfinite.

Message or message family	Meaning and correction
Cannot remove more statistics data…; Mean is undefined…	Check the statistical count and whether the bank was cleared. Repeated weights are not repeated observations.
Standard deviation/regression/estimation requires two data points	Enter at least two actual pairs.
Weights sum to zero; X/Y has zero variance; negative variance	Check weights, coordinate variation, mistaken removals, and cancellation in raw sums. Re-enter verified data at suitable precision when necessary.
No value for a financial register has been entered	An extended financial command needs an explicitly stored input, including a required zero. TVM's unwritten-zero convention is different.
Financial calculations require finite values and results	Check every participating input for NaN/infinity and an excessive or singular result.
Periodic interest must be greater than -100%	Rate is per period and must exceed the boundary; check rate units and sign.
Periods must be nonnegative…	Check n and the operation's need for positive or integral periods; cash-flow n is an index.
Opposite signs required; no finite solution; indeterminate result	Check cash signs and the complete input model. Some data has no solution; other data does not determine one unique answer.
Cash flows require CF0 and an integer N…	Start the list and store a valid final index n. CLfin removes n but leaves the list.
Cash-flow index/frequency/capacity error	Use an existing zero-based index, a positive integer repetition count, and at most 1000 expanded entries. Nj includes the occurrence already stored.
Invalid Gregorian date…	Check MDY/DMY, all six fractional positions, leap-year validity, and years 1–9999.
Bonds require…	Check settlement before maturity, coupon, clean-price basis, yield domain, and the required stored inputs.
Depreciation requires…	Check positive life, integer year, cost and salvage order, and a positive percentage factor for DB methods.
Financial solver: Root interval does not bracket a sign change	Choose appropriate IRR bounds or inspect the model; containing a touching root or two crossings can leave equal-sign endpoints.
Financial solver: Algorithm did not converge…	Check data, scales, and root selection. The work limit was reached; no usable solution was returned by that search.

Message or message family	Meaning and correction
Unable to save; the last saved state was restored	A storage commit failed. Inspect the restored working state before repeating any action.
Unable to restore calculator memory; Calculator memory is unavailable	The saved state could not be loaded. Preserve the error details and follow chapter 14; restarting is not a bank-clear command.

Some RPN operations return `NaN` (“not a number”) or $\pm\infty$ without a domain message. These are not finite answers. Check settings and operands; absence of a warning does not establish accuracy. See chapter 14 for recovery.

Appendix A

Learning and Mastering RPN

Learning RPN is learning to recognize the next small calculation inside a larger one. Supply its operands, perform it, and treat its result as one number. The stack keeps other unfinished parts available while you work. With practice, a long expression becomes a sequence of short, checkable steps rather than a string of symbols to memorize.

Chapters 2 and 3 introduce the controls and first calculations. This appendix develops the planning skill: translating notation, choosing an evaluation order, conserving stack space, reusing values, and checking intermediate results. Work through the examples slowly before trying to minimize keystrokes. A sequence you can explain is more useful than a shorter sequence whose operand order you cannot verify.

A.1 Infix, prefix, and postfix describe the same calculation

An operator can be written between its operands, before them, or after them. These positions give the names *infix*, *prefix*, and *postfix*. Ordinary written arithmetic is mostly infix: in $8 + 4$, the plus sign is between 8 and 4. RPN uses postfix: 8 4 + supplies both numbers before asking for their sum. Polish notation puts the operator first; *reverse* Polish notation puts it last.

Table A.1: Three notations for one expression. Spaces separate complete numbers or operators; these are mathematical descriptions, not literal calculator key sequences.

Notation	The quotient of $8 + 4$ and $7 - 3$
Infix	$(8 + 4)/(7 - 3)$
Prefix	$\div \quad + \quad 8 \quad 4 \quad - \quad 7 \quad 3$
Postfix	$8 \quad 4 \quad + \quad 7 \quad 3 \quad - \quad \div$

The postfix description reads as instructions: form 8 plus 4, form 7 minus 3, then divide the first result by the second. Each operator knows how many operands it needs. Addition takes two; square root takes one. Once an operation has used its operands, its result takes their place. The sequence itself therefore identifies the groups that infix notation marks with parentheses.

There is still mathematical grouping to respect. RPN removes the need to *enter* parentheses; it does not remove your responsibility to recognize them in the original problem. Likewise, there is no pending RPN multiplication waiting to take

precedence over an earlier addition. Each mathematical key acts immediately. You choose the order that implements the written expression.

SpicyRPN is a calculator, not a text reader for postfix formulas. Consecutive digits form one number, so typing 8 then 4 enters 84. Use **ENTER** to separate two newly typed numbers. By contrast, a function already finishes its numeric entry, so its result is ready for the next operation without another ENTER.

A.2 Prepare to see what each key does

Unless a sequence explicitly says otherwise, use RPN, real mode (**COMPLEX** absent), a fixed stack of 5, 600 working digits, and FIX 4. Show the Scientific panel. Angle units, date order, and payment timing do not affect this appendix's calculations. No example assumes empty memories; the memory examples explicitly replace the registers they use.

The shared procedure in section 2.2 establishes these settings. If they are already selected, cancel any active shift or pending command, then press **f** **CLEAR**, the orange function of $1/x$. The display should read `0.0000` with no **ENTRY** or pending-command flag. This is the *prepared start* below.

Repeat CLEAR before each independent experiment, but do not clear between rows described as successive steps. CLEAR leaves memories, settings, and the separate saved Last operands intact. With a command pending, a clear sequence can cancel that command instead of clearing the stack; first cancel pending commands as described in section 1.8.

Open the full-stack inspector between steps whenever you want to see where a waiting result went. Its small triangle is at the lower right of the display. Inspection does not commit a number or move a value. In this appendix, X means the displayed S0, and Y means S1. The traces put S0/X at the bottom and the deepest register at the top. Integers and fractions denote values; the actual FIX 4 display adds or rounds decimal places. An italic value is still being typed.

If number entry and waiting subtotals already feel familiar, continue with section A.4. The following short refresher is optional.

Optional refresher: when ENTER is needed

During entry, **ENTER** finishes the number in S0/X; when entry is inactive, it duplicates that value. A new number lifts the stack on its first character. Thus ENTER separates two freshly typed numbers, but is unnecessary between a completed result and a new number. An extra ENTER there leaves an unwanted duplicate deeper in the stack. Sections 2.3 and 3.5 provide worked examples.

A.3 Optional refresher: a waiting intermediate result

Think of each unfinished part as a value with a job. In $(8+4)/(7-3)$, the numerator's job is to wait while the denominator is calculated. The stack holds it in a deeper register while newer operands occupy X and Y. From the prepared start, press:

f CLEAR 8 ENTER 4 +

7 ENTER 3 − ÷

Table A.2: Three checkpoints for the quotient, in order. The numerator waits in S2 while both denominator operands are present.

Checkpoint	S0 (X)	S1 (Y)	Deeper needed value
Numerator ready	12	0	None
While typing 3	*3*	7	12 in S2
Denominator ready	4	12	None

The final division uses $Y/X = 12/4$, giving `3.0000`. Three useful values coexist at the busiest point; the padding zeros need no attention. Section 3.6 gives a full register-by-register trace of the same pattern. The next section shows how to choose that pattern from the written expression.

A.4 Translate grouping before translating keys

Begin with the last operation the written expression requires. A large fraction ends with division; a sum of two products ends with addition; a square root enclosing a sum ends with square root. Then plan the operands of that final operation. Repeat this process for each smaller part until every input is a number or a value you already have.

For $3 + 4 \times 5$, the final operation is addition: its operands are 3 and the product 20. For $(3 + 4) \times 5$, the final operation is multiplication: its operands are the sum 7 and 5. The numbers are the same, but their jobs and evaluation order differ.

Notice how two successive operators finish a nested calculation. In $20 - (6 - 2)$, the first subtraction leaves 4 in X and the waiting 20 in Y; the second subtraction obtains 16. In the power example, the first power obtains $3^2 = 9$, and the second computes 2^9. The power key always reads base Y and exponent X.

It is often helpful to mark completed subexpressions on paper. Replace $6 - 2$ mentally with a single box labeled 4, then read the remaining expression as $20 - 4$. An RPN result is exactly that kind of replacement, carried out on the stack.

Table A.3: Preserving written grouping. Start each row from the prepared RPN settings and f CLEAR. Results omit FIX 4's trailing zeros.

Expression	Keys after clearing	Result
$3+4\times5$	3 ENTER 4 ENTER 5 × +	23
$(3+4)\times5$	3 ENTER 4 + 5 ×	35
$(20-6)-2$	20 ENTER 6 − 2 −	12
$20-(6-2)$	20 ENTER 6 ENTER 2 − −	16
$2^{(3^2)}$	2 ENTER 3 ENTER 2 y^x y^x	512
$(2^3)^2$	2 ENTER 3 y^x 2 y^x	64

Single-operand functions finish their own groups

Apply a single-operand function only after its whole argument has been formed in X. Squaring one term and then adding another is different from adding both terms and squaring their sum. A fraction bar or a root sign can group an expression just as parentheses do.

Work outward from an inner difference

Use the prepared RPN start. Calculate

$$\frac{\sqrt{9+(12-4)^2}}{5}.$$

Square is the orange function of square root. First make the inner difference, square it, add 9, take the root, and finally divide:

f CLEAR 12 ENTER 4 − f x^2
9 + $\sqrt{x}$ 5 ÷

Checkpoints are 8, 64, 73, and $\sqrt{73}$; the final result is `1.7088`. Its retained value begins 1.7088007490635. Every step needs at most two useful stack values.

A.5 Choose an order that fits the stack

The length of a formula does not by itself determine its stack needs. A hundred successive additions can keep only an accumulated sum and one new number. A much shorter expression can need several waiting operands if you enter every outer value before evaluating its nested parts.

A fixed five-register stack can hold five useful values at once. Starting a sixth number discards the previous S4 immediately, on the first entry character. There is

no overflow prompt that asks whether that deepest value is still needed. A later drop repeats the new deepest value; it cannot recover the discarded one. Plan space before the lift.

One practical rule is to evaluate the more demanding part first. If a completed left part must wait while a right part uses four positions, you need five positions in total. If the right part is evaluated first, and the left part needs only one position, the second phase needs only two positions. Use SWAP before the final operation when this reverses the required operand order.

For a systematic count, suppose separate evaluations of the left and right parts need peaks of L and R useful values. Evaluating left first needs $\max(L, 1 + R)$; evaluating right first needs $\max(R, 1 + L)$. Choose the smaller count, allowing for the final operand order. A number needs one position, a unary function adds none, and two equally demanding parts need one extra position when combined. These counts describe ordinary arithmetic, without deliberate duplicates or retained-base functions such as percent.

Six nested operands with only two useful positions

Use the prepared RPN start. Evaluate the right-nested quotient

$$\cfrac{10}{\cfrac{20}{\cfrac{30}{\cfrac{40}{\cfrac{50}{60}}}}}.$$

Start with the innermost division, then introduce each outer numerator:

f CLEAR 50 ENTER 60 ÷
40 $x \leftrightarrow y$ ÷ 30 $x \leftrightarrow y$ ÷
20 $x \leftrightarrow y$ ÷ 10 $x \leftrightarrow y$ ÷

The successive mathematical quotients are 5/6, 48, 5/8, 32, and 5/16. The final display is `0.3125`. Each new outer numerator initially occupies X, so SWAP places it in Y before division by the waiting denominator.

Entering 10, 20, 30, 40, 50, and 60 before doing any arithmetic would require six positions. The following deliberately incorrect alternative overflows the prepared five-register stack and loses the initial 10, producing the incorrect answer `0.6250`:

f CLEAR 10 ENTER 20 ENTER 30 ENTER
40 ENTER 50 ENTER 60 ÷ ÷ ÷ ÷ ÷

The 10 was lost when 60 began. Repeated deepest-register copies later supply a second 20 in its place. Inspecting only the last division would not reveal how the wrong operand arose.

You can instead choose a larger fixed depth or an unlimited stack with STACK, as explained in section 2.10. Unlimited avoids the fixed-capacity loss but does not supply deepest-register repetition or roll. Another option is ordinary memory. None of these choices removes the need to know what each waiting value represents.

Changing the order in which two independent parts are calculated need not change their internal formulas. By contrast, regrouping a chain of additions or rewriting a quotient algebraically can change its finite-precision rounding. Use a convenient order while retaining the intended grouping, and consult chapter 5 for sensitive calculations.

A.6 The project website's expression equal to 42

The project's original online RPN primer includes the following expression.[1] It combines nested fractions, powers, a square root, and a product of two grouped expressions:

$$\frac{5^7 - \dfrac{24^3 + 66}{\sqrt{57 - 3 \times 7}}}{(2 \times 4 + 2)(6 + 3(5 \times 3 - 5)) + 1} = 42. \tag{A.1}$$

Here "complex expression" means intricate grouping; all its operands and results are real. Leave complex mode off. The walkthrough below uses the current SpicyRPN entry and stack rules.

Name the main intermediate results before entering them:

$$A = \sqrt{57 - 3 \times 7} = 6, \qquad B = \frac{24^3 + 66}{A} = 2315,$$
$$N = \frac{5^7 - B}{5} = 15162, \qquad C = 6 + 3(5 \times 3 - 5) = 36,$$
$$D = (2 \times 4 + 2)C + 1 = 361.$$

The last step is N/D. These letters are labels for the explanation, not memories to store or symbols to enter. There are two main phases: produce N, then keep it waiting while producing D.

[1] SpicyRPN, "RPN Primer," section "A More Complex Example," `https://spicyrpn.com/rpn-primer/`, consulted 3 October 2026. The equation and its illustrated keys use $57 - 3 \times 7$ under the square root; that is the expression transcribed here. The website describes an earlier application.

Build the entire numerator

Use the prepared real RPN start, fixed depth 5, 600 working digits, FIX 4, and no active shift or pending command. Follow all rows successively:

Keys	X after the row
f CLEAR 57 ENTER 3 ENTER 7 × — $\sqrt{x}$	6
24 ENTER 3 y^x 66 +	13890
$x \leftrightarrow y$ ÷	2315
5 ENTER 7 y^x	78125
$x \leftrightarrow y$ — 5 ÷	15162

The first row makes A. In the second row, A waits in Y while X becomes 13890; SWAP arranges 13890/6. Next B waits while 5^7 is formed. SWAP puts 5^7 in Y and B in X; subtraction gives 75810, and division by 5 produces N. The display is `15,162.0000`. Continue below without clearing or pressing an extra ENTER.

Build the denominator and join the two parts

Continue directly from the preceding numerator, with 15162 in X, all deeper positions zero, the same RPN settings, and no active shift or pending command. Do not clear between these rows:

Keys	X after the row
5 ENTER 3 × 5 —	10
3 × 6 +	36
2 ENTER 4 × 2 +	10
× 1 +	361
÷	42

The first two rows make C while N waits. The third makes the other factor, $2 \times 4 + 2$. Multiplication joins the factors; adding 1 completes D. The final division uses Y = 15162 and X = 361, displaying `42.0000`.

The stack reaches its largest useful population when 4 is being entered for the denominator's left factor: N, C, 2, and 4 coexist. Four positions are sufficient for this sequence; the default fifth position provides spare room. No ordinary memory is required.

An independent arithmetic check is particularly simple: $24^3+66 = 13890$, $13890/6 = 2315$, and $(78125 - 2315)/5 = 15162$. The denominator is $10 \times 36 + 1 = 361$, and $361 \times 42 = 15162$. These checks isolate the structure of the calculation; a correct final 42 is more convincing when its numerator and denominator also agree.

Table A.4: Checkpoints in the extended example, in chronological order. "Enter 4" is the busiest point in the denominator phase. Only N and D remain relevant before the final division.

Position	A ready	B ready	Enter 4	D ready	Finished
S4	0	0	0	0	0
S3	0	0	15162	0	0
S2	0	0	36	0	0
Y/S1	0	0	2	15162	0
X/S0	6	2315	*4*	361	42

A.7 Use deepest-register repetition deliberately

An ordinary binary operation drops a fixed stack by one position and repeats its deepest value. If every position above X contains the same factor, each multiplication exposes another copy in Y while S4 renews the supply. This can produce successive powers or repeated growth without retyping the factor.

The setup matters. A freshly typed value needs one ENTER to finish entry and four more to fill a five-register stack with copies. A value already produced by an operation needs only four ENTER presses, because its entry is already finished. A larger fixed depth needs more copies; an unlimited stack never renews a deepest register in this way.

Generate successive powers of two

Use the prepared RPN start with fixed depth exactly 5. Fill the stack:

f CLEAR 2 ENTER ENTER ENTER ENTER ENTER

All five positions now contain 2. Press × four times, inspecting after each if desired. X displays, successively, `4.0000`, `8.0000`, `16.0000`, and `32.0000`. Y, S2, S3, and S4 remain 2 after every multiplication.

Table A.5: The repeating factor in a fixed five-register stack. No new number is entered between these multiplications.

Position	Filled	First ×	Second ×	Third ×
S4	2	2	2	2
S3	2	2	2	2
S2	2	2	2	2
Y/S1	2	2	2	2
X/S0	2	4	8	16

This is a reusable operand, not a special repeat-operation mode. Pressing addition with the same prepared stack would repeatedly add 2 instead. Division calculates Y/X, so repeated division would not divide the displayed running result by 2 in the same way. Attend to operand order even when the repeated value is convenient.

Follow three periods of one-percent growth

Use the prepared RPN start with fixed depth 5. Form the factor $1 + 1/100 = 1.01$, then fill all positions with it:

f CLEAR 1 ENTER 100 ÷ 1 + ENTER ENTER ENTER ENTER

Now enter 1000 ENTER. X is 1000 and the other four positions contain 1.01. Press × once per period. The first three displays are `1,010.0000`, `1,020.1000`, and `1,030.3010`, agreeing with $1000(1.01)^j$ for $j = 1, 2, 3$. The four waiting factors remain 1.01.

This example uses ordinary multiplication, not the financial registers. It neither changes n nor counts periods for you, and it does not round each period's value to cents. Keep track of how many multiplications you performed. Use TVM or AMORT when you need their financial models and register updates; see chapter 11.

A new digit, a recall, a swap, or an unrelated function can disturb the prepared pattern. Inspect it before relying on another repeated factor, and rebuild it when changing tasks. The supply is replenished by the fixed-stack rule; the arithmetic still has finite precision and range.

A.8 Reverse subtraction or division safely

Evaluating the right-hand part first can save space. It also leaves that result in Y when the left-hand part arrives in X. For subtraction, division, powers, and indexed roots, put the operands in their intended positions before using the key. SWAP, the unshifted $x \leftrightarrow y$, exchanges only X and Y.

For finite real values, subtraction followed by sign change reverses the difference: $-(a - b) = b - a$. Division followed by reciprocal reverses a quotient when both operands and the intermediate quotient are nonzero and usable: $1/(a/b) = b/a$. The reciprocal method introduces an extra arithmetic operation and can encounter an invalid or extreme intermediate value. SWAP avoids forming the wrong quotient in the first place.

There is no comparable general one-key reversal for a^b and b^a. Use SWAP before power. The same discipline applies to a root's radicand and index. Remember also that sign change or reciprocal is a new unary calculation and updates the saved

Table A.6: Two ways to correct deliberately reversed operands. Start each independent row with the prepared RPN settings, no active shift or pending command, and f CLEAR.

Intended result	Keys after clearing	Result
$12 - 3$	3 ENTER 12 $x \leftrightarrow y$ $-$	9
$12 - 3$	3 ENTER 12 $-$ $+/-$	9
$12/3$	3 ENTER 12 $x \leftrightarrow y$ $\div$	4
$12/3$	3 ENTER 12 $\div$ $1/x$	4

operands. If your plan depends on LAST X or LAST Y, another calculation may have replaced the pair you intended to recover.

A.9 Give a reusable value an ordinary memory register

Use the stack for values needed soon in the current calculation. Use ordinary memory for a value that several calculations need, a result you want to preserve while clearing or rearranging the stack, or a checkpoint whose meaning would otherwise be hard to remember. Choosing a memory is a way to make the work easier to follow, not a failure to master the stack.

STO followed by a digit copies X into that register without removing X. RCL followed by a digit lifts the saved value onto the stack. Recall finishes a number being typed, so ENTER is unnecessary between that number and RCL. Ordinary recall also updates Last X; do not rely on an older saved operand after recalling a memory. Chapter 6 covers the full bank and clearing rules.

Reuse a circular cross-sectional area

Use the prepared RPN start. This example replaces R2; preserve its old value first if needed. For radius 1.25, calculate and store the area $A = \pi(1.25)^2$. Orange 0 supplies π:

f CLEAR 1.25 f x^2 f 0 $\times$ STO 2

The display is `4.9087`. R2 retains the working-precision area, not the four-place readout. For cylinders of lengths 3 and 8 in the same length unit, try each row:

Complete sequence	Volume
f CLEAR 3 RCL 2 $\times$	`14.7262`
f CLEAR 8 RCL 2 $\times$	`39.2699`

The results are $75\pi/16$ and $25\pi/2$, in cubic length units. R2 remains available after both calculations.

For a one-use intermediate, you can store it, clear the stack, calculate the other part, then recall it. Check the resulting order: that recall places the stored part in X. Use SWAP if it belongs in Y for the final operation. Writing a short note such as "R2 = area" is often more helpful than remembering the register number alone.

A.10 Practice, check, and recover

For each exercise, use the prepared real RPN settings, fixed depth 5, 600 working digits, FIX 4, and no active shift or pending command. Begin with f CLEAR. If an exercise specifies memory, initialize it as requested regardless of its earlier contents. Do not look at the answers until you have identified the final operation and the maximum number of useful values your sequence keeps at once.

1. Write $6(2+5)$ in postfix notation, then enter it.
2. Calculate $(17-5)/(2+4)$. Identify X and Y immediately before the final operation.
3. Calculate $(3^2+4^2)/(7-2)$, using the square function.
4. Calculate $9/[2+(8-5)^2]$, forming the denominator first.
5. Calculate both $2^{(1+3)}$ and $(2+1)^3$. Explain why they require different operation orders.
6. Calculate $1/[1+1/(1+1/2)]$, starting at the innermost fraction and using reciprocal.
7. Calculate $(20-3)/(7-2)$, forming $7-2$ first. Trace where the denominator waits.
8. Fill the five-register stack with 1.02, enter 200, and find the value after three successive multiplications. How many ENTER presses finish and fill a freshly typed factor?
9. Store $\sqrt{5^2+12^2}$ in R3, deliberately replacing its previous contents. Clear the stack and use two recalls to calculate $(R3+7)/(R3-5)$. R3 in this expression means the stored value, not a variable-name entry feature.
10. Enter and accept 96 ENTER 6 ENTER. Replace the accepted 6 with 8 without re-entering 96, then divide.
11. Intend to calculate $(7+5)+(48/6)$, but deliberately press multiplication for the division. Recover immediately with DROP, LAST Y, and LAST X, preserving the waiting subtotal 12, then complete the intended calculation.
12. Calculate $(2+3)((4+5)/(6+7))$. Show a checkpoint at its maximum useful stack population and explain why no memory is required.

A.11 Answers and annotated checkpoints

The sequences below follow the same prepared settings. Each begins after a fresh f CLEAR, unless an explicit later CLEAR appears. Line breaks do not mean ENTER

or CLEAR. The numeric results are given in FIX 4; the explanatory fractions are exact mathematical checks, not instructions to retype rounded values.

1. One group and a multiplier

One postfix form is 2 5 + 6 ×. Enter 2 ENTER 5 + 6 × to obtain 42.0000. Addition first makes 7; multiplication uses Y = 7 and X = 6. The equally valid 6 2 5 + × keeps an extra waiting value and requires three useful positions instead of two.

2. Two differences or sums become one quotient

Press 17 ENTER 5 — 2 ENTER 4 + ÷. The answer is 2.0000, from 12/6.

Position	After first —	Enter 4	Before final ÷
S4	0	0	0
S3	0	0	0
S2	0	12	0
Y/S1	0	2	12
X/S0	12	*4*	6

The numerator is temporarily in S2. Addition drops the denominator's two operands into one result, bringing that numerator back to Y.

3. Two squares, then a denominator

Press 3 f x^2 4 f x^2 + 7 ENTER 2 — ÷. The answer is 5.0000. The numerator is $9 + 16 = 25$; it waits while $7 - 2 = 5$ is formed. Typing 4 after the first square starts a new operand, so it needs no preceding ENTER.

4. A denominator before its numerator

Press 8 ENTER 5 — f x^2 2 + 9 $x \leftrightarrow y$ ÷. The answer is 0.8182, from 9/11. After typing 9, X is 9 and Y is 11. SWAP makes the numerator Y and denominator X before division.

5. Different arguments for power

For $2^{(1+3)}$, press 1 **ENTER** 3 + 2 $x \leftrightarrow y$ y^x. The answer is 16.0000. This sequence calculates the exponent first, then swaps to put base 2 in Y and exponent 4 in X.

For $(2+1)^3$, start again with CLEAR, then press 2 **ENTER** 1 + 3 y^x. The answer is 27.0000. Here the addition makes the base, so no swap is needed after entering exponent 3.

6. Repeated reciprocal around completed groups

Press 2 $1/x$ 1 + $1/x$ 1 + $1/x$. The mathematical checkpoints are 1/2, 3/2, 2/3, 5/3, and 3/5. The final display is 0.6000. Each reciprocal acts on the whole value then in X; no parentheses need to be entered.

7. Keep the earlier denominator in reserve

Press 7 **ENTER** 2 − 20 **ENTER** 3 − $x \leftrightarrow y$ ÷. The result is 3.4000, from 17/5.

Position	Denominator	Enter 3	Numerator	After SWAP
S4	0	0	0	0
S3	0	0	0	0
S2	0	5	0	0
Y/S1	0	20	5	17
X/S0	5	*3*	17	5

The denominator's position changes, but its value remains 5 until the final division consumes it.

8. Fill the fixed stack, then count three operations

Press 1.02 **ENTER** **ENTER** **ENTER** **ENTER** **ENTER**, then 200 **ENTER** × × ×. Five ENTER presses finish and fill the freshly typed factor. The running values are 204, 208.08, and 212.2416; the last display is 212.2416. Each multiplication retains 1.02 in all four deeper positions. The independent check is $200(1.02)^3$.

9. Initialize and reuse a memory

Press 5 f x^2 12 f x^2 + $\sqrt{x}$ STO 3. The value in X and R3 is 13, from $\sqrt{25+144}$. Then press:

f CLEAR RCL 3 7 + RCL 3 5 — ÷

The first sum is 20. The second recall places 13 in X and moves that sum to Y; typing 5 leaves the useful stack, from deeper to shallower, as 20, 13, 5. Subtraction produces X = 8 and Y = 20. Division gives `2.5000`, and R3 still contains 13.

10. Remove the accepted operand before replacing it

DROP is the orange function of roll down. The complete sequence after CLEAR is 96 ENTER 6 ENTER f DROP 8 ÷. DROP removes 6 and exposes 96 in X. Typing 8 lifts 96 back into Y, so division gives `12.0000`. CLx would leave a zero position in place; typing 8 afterward would put that zero in Y instead of 96.

11. Recover operands before another calculation replaces them

Reproduce the mistaken operation with 7 ENTER 5 + 48 ENTER 6 ×. S0/X is 288 and S1/Y is the waiting 12. The separate saved operands are Last Y = 48 and Last X = 6. LAST Y is blue $1/x$ and LAST X is blue $x \leftrightarrow y$. Immediately press:

f DROP g LAST Y g LAST X ÷ +

The repaired division yields 8; addition to the waiting 12 produces `20.0000`.

Position	Mistake	After DROP	Both recalls	After ÷
S4	0	0	0	0
S3	0	0	0	0
S2	0	0	12	0
Y/S1	12	0	48	12
X/S0	288	12	6	8

DROP and both recalls preserve the saved pair until the corrected division uses it. CLx would replace Last X with 288. LAST X/Y are operand recovery, not a restoration of the whole earlier stack or an undo of memory changes; see section 2.9.

12. Four useful values at the busiest point

Press 2 **ENTER** 3 + 4 **ENTER** 5 + 6 **ENTER** 7 + ÷ ×. The display is `3.4615`, from $5(9/13) = 45/13$.

While 7 is being entered, S4 is zero, S3 holds 5, S2 holds 9, Y holds 6, and X is the entry 7. These four useful values fit in the five-register stack. Addition leaves 13 in X, 9 in Y, and 5 in S2; division leaves 9/13 in X and 5 in Y; multiplication finishes the expression. This trace identifies why each deeper value was retained and exactly when it became an operand.

Developing a reliable habit

When practicing, pause at meaningful intermediate results instead of checking only the final display. Identify the first checkpoint that differs, then inspect the entry state and operand order there. Repeating the same mistaken sequence faster is unlikely to help. Trying another valid evaluation order, checking a simple limiting case, or substituting the result back into the original relation provides a more independent check.

Read the function's stack rules before extending this planning method to a new domain. Percent retains its base in Y; statistics can return two values; financial operations may use stored registers rather than the displayed operands. The ordinary consume-two, produce-one model used for most of this appendix is a strong starting point, but those specialized commands have the distinct effects documented in chapters 10 to 13.

Appendix B

Why RPN?

RPN makes a calculation a sequence of decisions about values: take these numbers, perform this operation, inspect the result, and continue. That can be a comfortable way to work when a calculation develops in stages. ALG makes the written expression the organizing structure. That can be more comfortable when the main task is to transcribe a formula and check that its grouping is correct.

Both approaches can express the same arithmetic. The useful question is which one helps you keep track of the work in front of you. This appendix explains RPN's practical appeal, its costs, and the evidence behind some of its reputation. For instruction in the actual keys, see chapters A, 3 and 4.

In the RPN examples, X is the main value, labeled S0 in the full-stack view; Y is the next position, labeled S1 there. These names identify current stack positions; the separate Last X and Last Y registers save earlier operands for recovery.

B.1 A calculation built from intermediate results

Suppose a calculation has two subtotals that must eventually be multiplied. With RPN, you calculate the first subtotal and leave it on the stack. You then calculate the second while retaining the first. Multiplication consumes those two results. The stack supplies temporary storage without requiring you to name a memory register for every partial answer.

This is the practical meaning of putting an operator *after* its operands. When you press an ordinary arithmetic operator, the values it needs are already present. You do not have to ask whether a multiplication is still pending or whether a later parenthesis will change which addition is performed first. You do have to know which values are currently in X and Y.

Two subtotals, then one multiplication

Calculate

$$(2 + 3)(4 + 5) = 45.$$

Select RPN, real mode, the default five-register stack, and FIX 4. Use the default working precision of 600 digits; the angle setting does not affect this example. With no pending command or active shift, clear the stack using f CLEAR, the orange function of the reciprocal key, as described in section 2.8.

1. Press 2 ENTER 3 +. The first subtotal is 5.0000 in X.
2. Press 4 ENTER 5 +. The second subtotal is 9.0000 in X; the first remains 5.0000 in Y.
3. Press ×. The answer is 45.0000 in X.

There is no extra ENTER between the first addition and the next number. Starting that number lifts the existing subtotal automatically.

The ability to stop at meaningful intermediate results is useful beyond textbook expressions. You might first obtain an area, then multiply by a rate, then add a fixed amount. At each stage you can ask whether the units, sign, and approximate size make sense. RPN supports that habit directly; it does not perform the reasoning for you. A plausible-looking subtotal can still come from the wrong inputs.

The stack also has a cost: its values are not labelled "area," "rate," or "subtotal." Their roles remain yours to remember. The full-stack view shows the numbers, not the formula that produced them. For values that must survive unrelated work, use the memories in chapter 6 rather than relying on their current positions.

B.2 What ALG offers in return

ALG keeps a different kind of context visible: the expression itself. In SpicyRPN's real and complex ALG displays, you can compare that expression with the formula you intended to enter. Parentheses record grouping, and precedence determines which operations are performed first. SpicyRPN's ALG mode is an expression-entry system; it should not be confused with every calculator that happens to have an equals key. The precise rules are in chapter 4.

For the preceding example, keep real mode and FIX 4. Select ALG; with no pending command or active shift, use its unshifted CLEAR. The full sequence

(2 + 3) × (4 + 5) =

also produces 45.0000. SpicyRPN additionally accepts adjacent parenthesized groups as multiplication, so the explicit × can be omitted in this particular sequence.

Table B.1: Different kinds of context in the two entry modes

Question	RPN	ALG
What records the plan?	Your sequence of operations and the values waiting on the stack.	The expression, its parentheses, and the precedence rules.
What can I inspect?	Current intermediate values, including deeper values in the stack inspector.	The entered expression in real or complex mode, plus the current value or completed answer. Worksheets retain their labeled results.
How do I continue?	Use the result already in X, bringing in more operands as needed.	Continue from a completed result with an operator, or insert **ANS** into a new expression.
What must I remember?	Operand order, stack positions, and when a value must be preserved.	Grouping, precedence, and where a function's argument begins and ends.
How do I correct work?	Edit the current number before operating; recover available operands or reconstruct an affected step afterward.	Edit the expression from its right edge. SpicyRPN does not provide arbitrary cursor placement within it.
What is retained?	The current stack and working state, rather than a browsable record of earlier calculations.	The current expression and answer state, rather than a browsable collection of earlier expressions.

The choice is not a choice between different mathematical laws. SpicyRPN uses the same underlying numerical operations for ordinary arithmetic in both modes. RPN does not acquire extra numerical accuracy merely by omitting parentheses, nor does ALG lose accuracy merely by using them. Working precision and the actual sequence of operations matter. Reordering an expression can change accumulated rounding even when the rearranged expression is mathematically equivalent; see chapters C and 5.

Financial calculations, statistics, and associative memories are available in both modes, with shared banks and formulas. RPN supplies operands from the stack and returns results there. ALG uses explicit computation and labeled worksheets, keeping multiple outputs available without a stack. For TVM, ALG financial keys always store the displayed operand, while **CPT** followed by a register solves for it; RPN follows the store-or-solve rules in chapter 11. These entry conventions, alongside the shared scientific functions, ordinary memories, and conversions, let you choose a working method without changing the underlying mathematics.

B.3 Counting keys fairly

RPN often saves grouping keys, but it may need ENTER to separate successive numbers. ALG may avoid parentheses by using precedence or implicit multiplication. Neither mode wins every key-count comparison.

Table B.2 counts the actual calculation keys in SpicyRPN: each digit, operator, parenthesis, ENTER, and = counts once. Mode selection and the initial clear are excluded in both modes. Each calculation starts independently with no pending command or active shift, real mode, FIX 4, and the default working precision; RPN uses a cleared default five-register stack, and ALG uses a cleared expression. No stored values, previous answer, or repeated-equals shortcut is assumed.

Expression	Result	RPN keys	ALG keys
$2+3$	5	4	4
$2+3\times4$, operands in written order	14	7	6
$(2+3)(4+5)$	45	9	11

Table B.2: Small comparisons, including cases that do not favor RPN

For the first row, RPN uses 2 ENTER 3 +; ALG uses 2 + 3 =. The separator replaces the equals key, so the totals are equal.

For the second row, preserving the written order of the numbers gives the RPN sequence

2 ENTER 3 ENTER 4 × +.

ALG uses 2 + 3 × 4 =, letting multiplication precedence do the grouping. An RPN user can instead calculate the product first:

3 ENTER 4 × 2 +.

That also takes six keys and gives `14.0000`. It requires choosing a different order of entry, which is useful work to recognize when comparing speed.

For the last row, the RPN sequence is the worked example's nine keys. ALG takes eleven using adjacent groups, or twelve using the explicit multiplication shown earlier. Counting unnecessary parentheses or an unnecessary multiplication would exaggerate RPN's advantage.

A key count measures only one part of a task. Reading a formula, deciding on an order, checking a subtotal, locating a shifted function, and repairing a mistake all take time. The keys also differ in size and location on a touch screen. A shorter sequence can therefore take longer to execute confidently. Conversely, a few extra keys can be worthwhile if they make the calculation easier to check.

B.4 Learning effort and opportunities for error

The basic RPN rule is short: supply the operands, then the operation. Fluent use requires more than memorizing that sentence. You learn to see an expression as a set of small calculations and to recognize which intermediate values can safely wait. That effort is most visible at the beginning, when translating a familiar written formula into a new entry order still takes conscious thought.

Some errors become less likely because RPN does not ask you to enter parentheses or defer ordinary arithmetic according to precedence. Other errors become possible because stack position carries meaning. The most useful comparison is therefore between concrete failure modes:

Operand order.
: Addition can hide a reversal because $a + b = b + a$. Subtraction and division cannot. In RPN, an ordinary binary operation uses Y op X. In ALG, the corresponding order appears left to right in the expression. When learning, include subtraction and division rather than practising only commutative operations.

Separating numbers.
: In RPN, ENTER separates consecutive numeric entries. A function already ends the current entry, so another ENTER is usually unnecessary before the next number. In SpicyRPN, the first ENTER during entry commits that number; an idle ENTER duplicates X. Confusing the two changes the stack. Section 2.3 explains the distinction.

Keeping unfinished work.
: A fixed stack can lose its deepest value when a new value is pushed onto a full set of useful registers. A larger or unlimited stack can remove that particular capacity constraint, but cannot identify the meaning of forgotten values. ALG instead asks you to keep track of open groups and the expression's right edge.

Recovering after an operation.
: RPN's LAST X and LAST Y can recover operands in the circumstances described in section 2.9. They are not an undo history and do not restore every affected register or stack position. ALG editing can repair an expression before evaluation, but reaching an earlier mistake may require removing and re-entering later material.

Neither notation prevents a transposed digit, the wrong angle unit, a misread percentage, or a cash flow with the wrong sign. Those are errors in the input or interpretation, even when the calculator follows the entry perfectly. Checking dimensions and estimating the answer remain useful in either mode.

A reasonable learning goal is to make a short calculation understandable at every step. Start with two operands, then a running subtotal, then two independent subtotals. Only afterward work on reducing stack depth or saving individual key

presses. The exercises in chapter A follow that progression. They also separate a correct, readable solution from an unusually compact one; the latter is not a prerequisite for using RPN well.

B.5 What the historical studies establish

RPN's reputation is not based solely on personal preference. Researchers have compared calculator entry systems experimentally. Three early papers are relevant, although their results need to be kept within the conditions they studied. The descriptions here are limited to the authors' published abstracts, rather than a new review of their full experimental data. The references and accessible records appear in section B.8.

Kasprzyk, Drury, and Bialas (1979) compared carefully matched calculators with users experienced in the respective entry systems. Their abstract reports results favoring RPN, while noting that the overall speed and accuracy differences were smaller than earlier findings. This supports an advantage in that comparison; it does not describe the entire learning curve of a person meeting RPN for the first time (source 1).

Agate and Drury (1980) reported faster calculation with the RPN calculator, particularly among participants with a technical background. Their abstract states that differences in the calculators' own speed and in participants' dexterity did not explain the measured difference. That is evidence about task performance, rather than simply faster arithmetic hardware (source 2).

Kreifeldt (1981) examined interrupted as well as uninterrupted work. His abstract reports no significant performance difference in the uninterrupted condition. Interruptions increased completion time for both groups, with a larger cost for ALG users. He also raises the possibility that users' approaches to formulating problems matched the calculator logic they had selected. This is a useful qualification: the relationship between the person, task, and interface may matter as much as a ranking of notations (source 3).

These reports should not be converted into a universal percentage saving or a promise that every user makes fewer mistakes with RPN. Calculator layout, prior experience, the expressions selected, opportunities to inspect work, and the way errors are scored all affect what a comparison means. A comparison of particular handheld calculators from that period also does not establish the same result for a modern phone interface or an expression editor with different correction facilities.

For choosing a working method, separate three questions: how quickly you can enter familiar calculations, how readily you can detect and repair mistakes, and how much practice you are willing to invest. The studies give a reason to take RPN seriously. Your own tasks determine whether its way of organizing intermediate results is helpful to you.

B.6 From Polish notation to pocket calculators

The name refers to the placement of operators. In ordinary *infix* notation, subtraction appears between its operands, as in $9 - 4$. In *prefix*, or Polish, notation the same operation is written $- \ 9 \ 4$. In *postfix*, or reverse Polish, notation it is written $9 \ 4 \ -$. "Reverse" concerns where the operator appears; it does not reverse the digits or interchange the two operands.

For operators with a known number of arguments, prefix and postfix notation can express grouping without parentheses. For example, $(2 + 3)(4 + 5)$ becomes the postfix sequence

$$2 \quad 3 \quad + \quad 4 \quad 5 \quad + \quad \times.$$

The calculator's **ENTER** keys separate typed numbers; they are part of the entry interface, rather than mathematical operators in that sequence. This distinction explains why calculators can share RPN notation while differing in their stack and entry rules.

The following are selected milestones, not a claim that a single machine or person accounts for the whole development.

The 1920s: Polish notation.
Jan Łukasiewicz developed prefix notation in mathematical logic. C. L. Hamblin's 1962 paper credits Łukasiewicz and cites work from 1921 and 1929. The purpose of that notation was broader than calculator keyboard design: it represented the structure of logical expressions without ordinary bracketing.

The 1950s and early 1960s: stack-based computation.
Hamblin's 1962 paper discusses reverse Polish notation as an order suitable for executing operations, refers to his 1957 addressless coding work, and connects the approach with the pushdown stores of the English Electric KDF9 and Burroughs B5000. The connection between postfix expressions and a stack therefore predates pocket scientific calculators (source 4).

1968: the HP 9100A.
Hewlett-Packard's desktop scientific calculator used RPN. It is an example of the notation moving into a calculator that users operated directly, before HP's pocket models (source 5).

1972: the HP-35.
HP introduced its handheld scientific calculator using RPN. Its contemporary *Hewlett-Packard Journal* explanation presents operands followed by operations and the stack as storage for intermediate results. It also describes evaluating an expression from its inner groups outward (source 5).

1981: the HP-12C.
HP introduced this programmable financial calculator with business functions including depreciation and date arithmetic (source 5).

The enduring idea is smaller than any particular keyboard: keep values available and apply an operation when its arguments are ready. Stack depth, duplication rules, memory facilities, and the display are separate design decisions. A user familiar with an older HP calculator should therefore learn SpicyRPN's entry rules explicitly; see section 2.11.

B.7 Why SpicyRPN takes this form

The original SpicyRPN website gives a personal account of the project's origin. Its author describes using an HP-15C from university onward and wanting an RPN calculator available on an iPhone. The site's RPN primer also explains two deliberate choices: a fifth stack position for times when four felt restrictive, and revised entry behavior to avoid an unnecessary push when confirming a number before a unary operation. Those are documented motivations for the original design, rather than motives inferred from the present feature list (source 6).

The current application retains a five-register default but allows a fixed depth from 2 to 1000, or an unlimited stack. It lifts the stack once when numeric entry begins; **ENTER** during that entry commits without a second lift. A further **ENTER** duplicates the committed value. These rules make the distinction between *finishing entry* and *requesting another copy* explicit. They are SpicyRPN's rules, not a claim of identical behavior across HP models.

Other current choices shape the same working experience:

- The full-stack inspector makes waiting intermediate values available for inspection without changing them. Ordinary and associative memories provide longer-lived storage when stack position is the wrong way to organize a value.
- Working precision is configurable, separately from the display. The default is 600 decimal digits of working precision, with supported settings from 16 to 10000. That is a numerical facility shared by the entry modes, not a consequence of RPN notation.
- RPN and ALG retain independent unfinished work. Changing modes does not translate the stack into an expression or evaluate the expression into the stack. Shared memories provide an explicit way to transfer a result when needed.
- The financial commands use dedicated registers and conventions, alongside the stack. Familiarity with RPN does not replace learning those conventions, including cash-flow signs, periodic interest rates, and the store-or-solve behavior described in chapter 11.

The capabilities and operating rules in this manual describe SpicyRPN Release 2.0.0, following the original application offered on the App Store around 2019. The historical account explains where the design began; the preceding chapters explain the calculator you are using.

B.8 Sources and further reading

The historical statements above use the following primary accounts. The three experimental summaries rely on author abstracts; no effect size, sample count, or modern-interface benchmark is inferred from them.

1. D. M. Kasprzyk, C. G. Drury, and W. F. Bialas, "Human behaviour and performance in calculator use with Algebraic and Reverse Polish Notation," *Ergonomics* 22(9), 1011–1019 (1979). DOI: 10.1080/00140137908924675. The SUNY research record reproduces the authors' abstract.
2. S. J. Agate and C. G. Drury, "Electronic calculators: which notation is the better?" *Applied Ergonomics* 11(1), 2–6 (1980). DOI: 10.1016/0003-6870(80)90114-3. The original abstract is available in PubMed, record 15676368.
3. John G. Kreifeldt, "Hand Calculator Performance Under Interrupted Operation," *Proceedings of the Human Factors Society Annual Meeting* 25(1), 329–332 (1981). DOI: 10.1177/107118138102500187. The publisher's abstract distinguishes uninterrupted performance from the cost of interruption.
4. C. L. Hamblin, "Translation to and from Polish Notation," *The Computer Journal* 5(3), 210–213 (1962). DOI: 10.1093/comjnl/5.3.210. This contemporary paper explains both forms of notation, their translation, and their relation to stack-based execution; its references identify the earlier Łukasiewicz and Hamblin works.
5. Hewlett-Packard's Virtual Museum entries for the 9100A (1968), HP-35 (1972), and HP-12C (1981); and the original sidebar "Reverse Polish Notation," *Hewlett-Packard Journal*, June 1972. These support the selected product milestones, rather than a claim that HP originated the underlying mathematical notation.
6. The original SpicyRPN RPN Primer, especially its account of the author's HP-15C and its explanation of SpicyRPN's five-position stack and entry behavior. Current capabilities are documented from the source version covered by this manual; chapter D explains obtaining that source.

Appendix C

Mathematical and Numerical Reference

This appendix defines the mathematics used by the current calculator and the numerical conventions that affect its results. The operating instructions and key sequences are in chapters 7 to 13. A formula here specifies the intended calculation; its individual operations still take place at finite working precision. Algebraically equivalent formulas need not produce identical last digits.

Unless a section says otherwise, the formulas below assume finite real arguments; individual commands may reject nonfinite inputs or propagate a nonfinite result. For a two-operand scientific command in RPN, y denotes the older operand in Y, displayed at S1, and x the newer operand in X, displayed at S0. Last X and Last Y are separately saved operands, not these current positions. In ALG, the corresponding operands are the left and right arguments; ALG has no S0/S1 stack positions. Other mathematical symbols name the quantities defined in their sections, with register or worksheet mappings given in the operating chapters. Both modes use the scientific, statistical, financial, and conversion mathematics described here. RPN supplies operands from the stack; ALG uses its expression and labeled worksheets. Their operating sequences and result placement are documented in chapter 15.

C.1 Finite multiple precision

Decimal settings and binary storage

SpicyRPN uses MPFR numbers with a binary significand. A requested working precision of d decimal digits selects

$$p = \lceil d \log_2 10 \rceil$$

binary significand bits. For example, 16, 600, and 10,000 decimal digits select 54, 1,994, and 33,220 bits. The default is 600 decimal digits; the settings interface accepts 16 through 10,000. Exponents have a separate, finite range. Increasing significand precision does not make the exponent range unlimited.

A finite decimal fraction is exact in binary only when its reduced denominator is a power of two and its significand fits. Thus a catalog factor such as 0.3048 can be an exact decimal definition while its stored binary value is rounded. Signed zero, infinities, and a not-a-number result are also represented. A mathematical domain

error may therefore appear as a nonfinite result, or a command may reject its inputs before evaluating the underlying function.

Precision propagation

Each stored real value retains the precision at which it was created. Changing the setting does not rewrite the stack, memories, or financial and statistical registers. Subsequent calculations create results at the current working precision, using the available bits of their operands. Raising precision cannot recover digits already lost when a value was entered or computed. A complex pair contains two such real components, which can have different creation precisions. See section 5.7.

The usual directly evaluated MPFR arithmetic and elementary functions use round-to-nearest, with an exact halfway case resolved toward an even final binary significand bit. MPFR treats the stored input values as exact when rounding one operation. This is a guarantee about that operation on those inputs, not about a sequence of operations, uncertain measurements, or a SpicyRPN iterative algorithm. MPFR's definitions are given in its manual, `https://www.mpfr.org/mpfr-4.2.2/mpfr.html#Rounding`.

Displayed digits and reliable digits

FIX, SCI, and ENG select formatting; they do not increase the information in a value. Display rounding uses nearest with ties to even. FIX may switch to scientific notation when fixed notation cannot usefully show the result, including a nonzero value that would otherwise display as zero. The permitted display setting is zero through the current working digit count. The meanings of the digit setting and the notation fallback are explained in section 5.3.

Cancellation, ill conditioning, repeated rounding, and numerical root finding can reduce the number of trustworthy digits. A display containing hundreds of digits is not an error bound. Repeating a calculation with freshly entered inputs at a higher precision is a useful check; agreement alone is not a proof of accuracy or uniqueness.

C.2 Rounding and integer conventions

For a real x, $\operatorname{trunc}(x)$ removes its fractional part toward zero, $\lfloor x \rfloor$ rounds down, and $\lceil x \rceil$ rounds up. The fractional-part function is

$$\operatorname{frac}(x) = x - \operatorname{trunc}(x).$$

It is negative for a negative noninteger input. NINT rounds to the nearest integer with ties to even: both 1.5 and 2.5 round to 2.

The financial RND command instead rounds the value to k decimal places, where k is the display digit setting. Its mathematical rule is

$$\mathrm{RND}_k(x) = \mathrm{sgn}(x)\,\frac{\left\lfloor |x|10^k + \frac{1}{2} \right\rfloor}{10^k}. \tag{C.1}$$

An exact halfway decimal case rounds away from zero. The scaling and rounding operations themselves use finite precision. RND uses decimal places even when SCI or ENG is selected; it changes the stored result. See section 5.6.

The two real remainder conventions differ in their quotient:

$$\mathrm{mod}(y, x) = y - \mathrm{trunc}(y/x)x,$$
$$\mathrm{rem}(y, x) = y - \mathrm{nint}(y/x)x.$$

For REM, the quotient is chosen by nearest-integer, ties-to-even rounding; these are definitions of the remainder operation, not instructions to round a displayed quotient first. The divisor must be nonzero. MOD consequently has the dividend's sign when its result is nonzero. It is not the always nonnegative modulo convention used by some programming languages.

C.3 Real scientific definitions and domains

Arithmetic, powers, and logarithms

The basic binary operations are $y + x$, $y - x$, yx, and y/x. Negation means $-x$, absolute value means $|x|$, reciprocal means $1/x$, square means x^2, and the real square root is the nonnegative root, defined for $x \geq 0$. The real power is y^x. A negative base requires an integer exponent for this real power command. A zero base with a negative exponent is not a finite result; the power implementation returns 1 for 0^0.

For the root command, y is the radicand and x the index. An integer index must be nonzero and lie between −1,000,000 and 1,000,000. An odd integer index permits a negative radicand; an even one does not. A negative index takes the reciprocal of the corresponding positive-index root. A noninteger index uses $y^{1/x}$ and requires a nonnegative real radicand. These real rules differ from the complex root convention in section C.4.

The exponential functions are e^x and 10^x. Natural and common logarithms are $\ln x$ and $\log_{10} x = \ln x / \ln 10$; the binary logarithm is $\log_2 x = \ln x / \ln 2$. They require positive real input for a finite result. The two-argument logarithm is

$$\log_y x = \frac{\ln x}{\ln y}, \qquad x > 0, \quad y > 0, \quad y \neq 1.$$

The constants π, $e = \exp(1)$, and $\ln 2$ are computed at the current working precision; they are not stored short decimal approximations.

The percentage functions use these operand orders:

$$\%: \frac{yx}{100}, \qquad \%\ \text{change}: \frac{100(x-y)}{y}, \qquad \%\ \text{total}: \frac{100x}{y}.$$

The latter two require $y \neq 0$. In RPN, their stack behavior preserves the base in Y; ALG treats them as binary functions of the left and right operands. See chapters 3 and 15.

Angles and hyperbolic functions

Let the conversion from the selected angle unit to radians be

$$s = \begin{cases} \pi/180 & \text{DEG}, \\ 1 & \text{RAD}, \\ \pi/200 & \text{GRAD}. \end{cases}$$

Circular functions evaluate $\sin(sx)$, $\cos(sx)$, and $\tan(sx)$. Their inverse results are the principal radian values divided by s. The real implementation uses MPFR's angle-unit operations where available, so the displayed formula does not imply a preliminary rounded multiplication by a stored value of s.

Table C.1: Real inverse functions: domains and principal ranges in radians.

Function	Input domain	Principal range
$\arcsin x$	$[-1, 1]$	$[-\pi/2, \pi/2]$
$\arccos x$	$[-1, 1]$	$[0, \pi]$
$\arctan x$	All finite real x	$(-\pi/2, \pi/2)$
$\operatorname{asinh} x$	All finite real x	All real values
$\operatorname{acosh} x$	$[1, \infty)$	$[0, \infty)$
$\operatorname{atanh} x$	$(-1, 1)$	All real values

The two-argument angle function uses $\operatorname{atan2}(y, x)$, with quadrant determined from both arguments, and returns the result in the selected angle unit. Its signed-zero cases follow the underlying real operation. The polar interpretation of complex numbers is described in chapter 8.

Hyperbolic functions do not use the angle setting:

$$\sinh x = \frac{e^x - e^{-x}}{2}, \qquad \cosh x = \frac{e^x + e^{-x}}{2}, \qquad \tanh x = \frac{\sinh x}{\cosh x}.$$

These identities define the functions; the implementation calls the numerical library rather than necessarily evaluating the displayed differences.

Factorials, combinatorics, and special functions

The ordinary factorial and permutation/combination commands accept integers:

$$n! = \prod_{j=1}^{n} j, \qquad 0! = 1,$$
$$ {}_nP_k = \frac{n!}{(n-k)!},$$
$$ {}_nC_k = \frac{n!}{k!(n-k)!}.$$

The bounds are $0 \le n \le 10{,}000$ and $0 \le k \le n$. Combinations use the smaller of k and $n-k$, with interleaved multiplication and division. The result is mathematically an integer, but it is stored at finite precision and need not retain every integer digit.

For positive real x,

$$\Gamma(x) = \int_0^\infty t^{x-1} e^{-t}\, dt, \qquad \Gamma(x+1) = x\Gamma(x).$$

Continuation defines the function elsewhere except its poles at zero and negative integers. The real logarithmic gamma function is the real $\ln \Gamma(x)$ where defined; it is not a request for $\ln |\Gamma(x)|$ on intervals where gamma is negative. The logarithmic factorial computes $\ln \Gamma(n+1)$ for a finite nonnegative integer n; it does not impose the ordinary factorial's 10,000 bound.

The error functions are

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2}\, dt, \qquad \operatorname{erfc}(x) = 1 - \operatorname{erf}(x).$$

For real arguments, ERFC is evaluated directly by the library, avoiding the loss that a separate subtraction from 1 can cause in a small tail. Complex implementations and their limits are described below.

The random-number command constructs a uniformly sampled decimal digit string at the requested working digit count and converts it to a stored number. A conversion that rounds to 1 is retried, so the stored result lies in $[0, 1)$. There is no user seed control. This is finite random sampling, not a supply of arbitrary exact real numbers.

C.4 Complex principal values

Write $z = a + ib$. MPC provides complex addition, subtraction, multiplication, division, negation, reciprocal, magnitude, powers, square root, exponentials, logarithms, and circular/hyperbolic functions and their inverses. The C bridge transfers

the stored MPFR components directly, without converting them through machine floating point or decimal text. Scalar coefficients and valid real-function paths use MPFR. RPN and ALG share this numerical implementation.

For a single MPC primitive, the real and imaginary results are correctly rounded independently to nearest, ties to even, at the precision of that call. Direct elementary operations request the selected working precision. This guarantee concerns the operation on its stored inputs, not the accuracy of those inputs or an entire composed calculation. See the MPC reference manual for the upstream definitions.

For polar magnitude ρ and displayed phase θ, the rectangular components are

$$a = \rho\cos(s\theta), \qquad b = \rho\sin(s\theta),$$

where s is the selected angle-unit scale from section C.3. In the reverse direction, $\rho = |z|$ and $\theta = \operatorname{atan2}(b, a)/s$, with the two-argument angle interpreted in radians before that scaling. The displayed phase of zero is set to zero. These representation changes do not redefine the logarithm branch below.

Composed indexed roots, arbitrary-base logarithms, and percentages use 20 additional decimal guard digits and round the final components back to the requested working precision. Angle-unit conversions and custom special functions also use additional intermediate precision as needed by their algorithms. A zero-imaginary input can use a real-library result when that result is not not-a-number. Consequently, a scalar special case may use a different numerical path from a nearby genuinely complex value. Guard digits reduce intermediate rounding; they do not guarantee correctly rounded final results for compositions, conversions, or approximations.

Logarithms, square roots, powers, and roots

The principal logarithm is

$$\operatorname{Log} z = \ln|z| + i\operatorname{Arg} z, \qquad -\pi < \operatorname{Arg} z \le \pi. \tag{C.2}$$

The negative real axis is taken on its upper side: for $r > 0$, $\operatorname{Log}(-r) = \ln r + i\pi$. A negative zero in the imaginary component does not choose the lower side of that axis. The imaginary part of a logarithm remains in radians in every angle mode.

The principal square root has nonnegative real part and, on the negative real axis, positive imaginary part:

$$\sqrt{-r} = i\sqrt{r} \quad (r > 0).$$

In the general case it is the principal value associated with $\exp(\frac{1}{2}\operatorname{Log} z)$; MPC evaluates square root directly.

The common and binary logarithms equal $\operatorname{Log} z/\ln 10$ and $\operatorname{Log} z/\ln 2$; MPC evaluates each directly. A logarithm of argument z to complex base w divides $\operatorname{Log} z$

by $\operatorname{Log} w$. Zeros of a denominator remain singular cases; complex mode does not supply a separate multivalued logarithm selection.

General complex powers and roots follow

$$z^w = \exp(w \operatorname{Log} z), \qquad \sqrt[w]{z} = z^{1/w}, \quad w \neq 0.$$

The power command uses MPC's power function directly, and the integer-power operation uses its integer-exponent primitive. The displayed identities define branch choices rather than a mandatory evaluation sequence. The implementation returns 1 for 0^0 and zero for zero raised to a positive real power. Other zero-base cases must be judged by their singularity rather than by the logarithm identity alone.

The complex root is one principal value, not a list of all roots. Thus the principal complex cube root of -8 is $1 + i\sqrt{3}$, whereas the real odd-root command gives -2 in real mode. See table 8.4 for representative branch-boundary values.

Circular and inverse functions

The complex exponential and circular functions satisfy

$$\begin{aligned} e^{a+ib} &= e^a(\cos b + i \sin b), \\ \sin(a+ib) &= \sin a \cosh b + i \cos a \sinh b, \\ \cos(a+ib) &= \cos a \cosh b - i \sin a \sinh b, \\ \tan z &= \sin z / \cos z. \end{aligned}$$

These are mathematical identities; MPC evaluates the functions directly, including tangent without a separate rounded sine/cosine quotient. Circular input is scaled to radians in *both* components according to the angle setting. Both components of an inverse circular result are scaled back to the selected angle unit. Hyperbolic functions and their inverses are not angle-scaled.

The principal inverse functions satisfy the following identities away from their singularities, with the exact-axis choices in table 8.4:

$$\begin{aligned} \arcsin z &= -i \operatorname{Log}\left(iz + \sqrt{1 - z^2}\right), \\ \arccos z &= \frac{\pi}{2} - \arcsin z, \\ \arctan z &= \frac{i}{2}\left[\operatorname{Log}(1 - iz) - \operatorname{Log}(1 + iz)\right], \\ \operatorname{acosh} z &= \operatorname{Log}\left(z + \sqrt{z+1}\sqrt{z-1}\right), \\ \operatorname{atanh} z &= \frac{1}{2}\left[\operatorname{Log}(1 + z) - \operatorname{Log}(1 - z)\right]. \end{aligned}$$

For inverse hyperbolic sine, define $H(z) = \operatorname{Log}(z + \sqrt{z^2 + 1})$. The branch selection follows $H(z)$ when $\Re z \geq 0$, and $-H(-z)$ when $\Re z < 0$. All logarithms and square

roots in these identities use the principal conventions above. MPC evaluates the six inverse functions directly, instead of forming these logarithmic differences or differences from $\pi/2$. The bridge deliberately chooses signed zeros on the axes to preserve SpicyRPN's documented branch values. Singularities and branch cuts remain part of the result; see section 8.9.

Complex gamma and error functions

MPC does not provide the following special functions. SpicyRPN's algorithms use MPC for complex intermediate arithmetic and elementary functions, with MPFR for real coefficients. Correctly rounded primitives do not certify the accuracy of a complete approximation or series.

Complex gamma uses a finite Spouge approximation, with the reflection identity

$$\Gamma(z) = \frac{\pi}{\sin(\pi z)\Gamma(1-z)}$$

when $\Re z < \frac{1}{2}$. At requested precision d, the coefficient count is $\lfloor 3d/2 \rfloor + 24$, and the calculation uses roughly $2d + 50$ decimal working digits before rounding back. The supported request is at most 10,000 digits. These choices are numerical safeguards rather than a certificate that every returned digit is correct, especially near poles. For a genuinely complex argument, logarithmic gamma is the principal logarithm of the computed gamma value, not a continuously unwrapped log-gamma branch. Complex factorial continues the real factorial as $z! = \Gamma(z+1)$, and complex logarithmic factorial uses $\operatorname{Log}\Gamma(z+1)$. These continuations also apply to real inputs for which the ordinary real factorial path is rejected; they are not restricted to nonnegative integers in complex mode. Poles remain excluded. A valid real fast path retains the real convention described above.

The complex error function uses the series

$$\operatorname{erf}(z) = \frac{2}{\sqrt{\pi}} \sum_{j=0}^{\infty} \frac{(-1)^j z^{2j+1}}{j!(2j+1)}.$$

It applies a magnitude limit of approximately $|z| \leq 100$, additional precision depending on $|z|^2$, and a finite term budget and relative-term stopping test. The preliminary magnitude check uses a coarse conversion to a machine number, so it is not an exact boundary test at radius 100. Failure to converge is reported rather than extending the series indefinitely. Complex ERFC is computed as $1 - \operatorname{erf}(z)$, so small complex tails can suffer cancellation. This differs from real ERFC's direct path.

Integer rounding, real remainders, the two-argument real angle, permutations, combinations, RND, statistics, financial functions, and unit conversions remain real operations. Turning complex mode off preserves existing stored imaginary components, but new calculations, memory writes, recalls, and register assignments

use only the real component and write zero imaginary part. A recall does not change its source memory. Associative keys always use only the real component, regardless of CMPLX; equal real keys identify one entry. See sections 6.5, 6.7 and 8.7. Complex mode does not turn every operation into a complex analytic continuation.

C.5 Statistical accumulators and results

The statistical bank stores a count n and five sums, not individual observations:

$$n, \quad \sum x, \quad \sum x^2, \quad \sum y, \quad \sum y^2, \quad \sum xy.$$

Each sum runs over the entered observation pairs. Here x, y are data coordinates, not names of stored stack positions; n is independent of the financial n register. The input and result placement in chapter 10 supplies the RPN or ALG mapping. Adding a pair adds $(1, x, x^2, y, y^2, xy)$; removing it subtracts the same quantities. A removal is blocked when it would make the count negative, but there is no sample list with which to establish that the pair was ever entered. Statistical quantities can therefore become inconsistent after an incorrect removal. Addition and removal require finite real components of both sample coordinates and finite existing and proposed values for all six totals. NaN, infinity, or overflow to a nonfinite total rejects the entire update without changing the bank. Imaginary components are not statistical inputs. These checks do not eliminate cancellation or validate the sample history. The R4–R9 transfer copies accumulators into ordinary memory; it is not a second sample bank or a reverse import mechanism.

The ordinary means and weighted mean are

$$\bar{x} = \frac{\sum x}{n}, \qquad \bar{y} = \frac{\sum y}{n}, \qquad \bar{x}_w = \frac{\sum xy}{\sum y}.$$

The ordinary means require $n > 0$. The weighted mean additionally requires $\sum y \neq 0$ and treats each y_j as a weight. In the other operations, y remains the second measured variable; the bank is not globally switched into a weighted-statistics mode.

Define the centered, count-scaled sums using the same M, N, P notation as section 10.7:

$$\begin{aligned} M &= n\sum x^2 - (\sum x)^2, \\ N &= n\sum y^2 - (\sum y)^2, \\ P &= n\sum xy - (\sum x)(\sum y). \end{aligned} \tag{C.3}$$

Sample standard deviations are

$$s_x = \sqrt{\frac{M}{n(n-1)}}, \qquad s_y = \sqrt{\frac{N}{n(n-1)}}, \qquad n > 1.$$

They use the sample divisor $n-1$, not the population divisor n. A negative variance numerator is invalid, even when it arose from rounding rather than from the intended data.

For the least-squares line $\hat{y} = a + bx$,

$$b = \frac{P}{M}, \qquad a = \frac{M\sum y - P\sum x}{nM}.$$

The fit requires $n > 1$ and $M > 0$. A constant y is permissible for the fitted line, but not for a correlation coefficient. The estimate and correlation command uses

$$\hat{y}(x_0) = \frac{(nx_0 - \sum x)P + M\sum y}{nM}, \qquad r = \frac{P}{\sqrt{MN}}. \tag{C.4}$$

That command requires $n > 1$, $M > 0$, and $N > 0$, because it returns both the estimate and correlation. Its estimate is evaluated by dividing the numerator by n, then by M. The correlation is not clamped to $[-1, 1]$ after calculation.

These are raw-sum formulas. When observations have a large common offset and a very small spread, the subtractions defining M, N, P can lose many digits. Higher precision or a carefully chosen change of origin can help. Operational examples and the order of returned results are in chapter 10.

C.6 Time value of money

Common equation and sign convention

Let n be the number of periods, $r = i/100$ the interest rate per period, $q = 1 + r$, and $b = 0$ for END or $b = 1$ for BEGIN. Money paid and money received must have opposite signs from one consistent viewpoint. Define

$$A_f = \begin{cases} q^b(q^n - 1)/r & r \neq 0, \\ n & r = 0, \end{cases} \qquad A_p = A_f q^{-n}.$$

The calculator solves the equation

$$\mathrm{PV}\, q^n + \mathrm{PMT}\, A_f + \mathrm{FV} = 0. \tag{C.5}$$

All four nontarget registers participate; an unwritten register supplies zero. The interest domain is $r > -1$, equivalent to $i > -100$. The supplied period count must be nonnegative, and solving PMT or interest requires a positive period count. All supplied values must be finite.

These are the same shared registers in both entry modes. In RPN, the financial key either stores or solves according to the entry state. In ALG, the register key stores the displayed operand, and **CPT** followed by the register key solves it. ALG's

other financial worksheets use their accepted named inputs, initialized from the corresponding registers or worksheet defaults. Accepting a linked field with ENTER writes that financial register; COMPUTE applies the calculation's stated updates. See chapter 11 for the full input and result conventions.

The direct solutions are

$$\begin{aligned} \mathrm{PV} &= -\left(\mathrm{FV}\, q^{-n} + \mathrm{PMT}\, A_p\right), \\ \mathrm{FV} &= -\left(\mathrm{PV}\, q^{n} + \mathrm{PMT}\, A_f\right), \\ \mathrm{PMT} &= -\frac{\mathrm{PV} + \mathrm{FV}\, q^{-n}}{A_p}. \end{aligned}$$

At zero interest these reduce to $\mathrm{PV} + n\mathrm{PMT} + \mathrm{FV} = 0$. A manually supplied fractional n is used in these compound-growth formulas. It does not activate an HP odd-first-period interest convention.

For improved behavior near zero interest, let $t = \ln(1 + r)$. For nonzero t, the annuity factors are evaluated using

$$\begin{aligned} H &= \frac{\sinh(nt/2)}{\sinh(t/2)}, \\ A_f &= H \exp\!\left(\frac{n-1}{2}t + bt\right), \\ A_p &= H \exp\!\left(-\frac{n+1}{2}t + bt\right). \end{aligned}$$

A small-argument logarithm uses $\ln(1+x) = 2\,\mathrm{atanh}(x/(2+x))$ for $|x| < 1/2$. These evaluation choices matter when r is tiny relative to the working precision.

Solving and rounding the number of periods

At $r = 0$, the unrounded solution is

$$n_* = -\frac{\mathrm{PV} + \mathrm{FV}}{\mathrm{PMT}}.$$

If PMT is zero, a zero numerator makes the equation indeterminate; otherwise there is no solution. For nonzero interest, define

$$D = r\mathrm{PV} + \mathrm{PMT}\, q^{b}, \qquad \delta = -\frac{r(\mathrm{PV} + \mathrm{FV})}{D}.$$

When $D \neq 0$ and $1 + \delta > 0$,

$$n_* = \frac{\ln(1+\delta)}{\ln(1+r)}.$$

A zero D is checked for an indeterminate or impossible equation. A negative or nonfinite n_* is rejected.

A solved n is then rounded with the financial period rule, rather than ordinary nearest-integer rounding. Let $h = \lfloor n_* \rfloor$, $f = n_* - h$, and let d be the working decimal precision. Define

$$\epsilon = \min\left(10^{-\max(1,d-2)} \max(1, n_*),\ 10^{-12}\right).$$

If $f < 0.005 - \epsilon$, the stored result is h; otherwise it is $h + 1$. The small guard makes a numerically computed threshold case round up. This rule applies to solving n, not to a fractional value manually stored in n.

Solving interest and interpreting the period

With PMT zero, the rate solution follows

$$t = \frac{\ln(-\mathrm{FV}/\mathrm{PV})}{n},$$

requiring opposite, nonzero PV and FV. When $n = 1$, a nonzero payment also permits a direct solution:

$$q = \begin{cases} -(\mathrm{FV} + \mathrm{PMT})/\mathrm{PV} & \text{END}, \\ -\mathrm{FV}/(\mathrm{PV} + \mathrm{PMT}) & \text{BEGIN}. \end{cases}$$

A zero denominator is classified as indeterminate or impossible, and a permissible solution requires $q > 0$. Other cases use numerical root finding in $t = \ln q$, starting at zero; see section C.9. The final periodic percentage is evaluated as

$$i = 200e^{t/2} \sinh(t/2),$$

which is mathematically $100(e^t - 1)$.

A period is the common unit of n, i, and PMT. The annual-rate shortcut divides a nominal annual percentage by 12; the years shortcut multiplies a year count by 12. Neither converts an effective annual rate. For an effective annual fraction j and m equal periods per year, the equivalent periodic fraction would be $(1 + j)^{1/m} - 1$. See chapter 11 for complete setups and cash-sign examples.

C.7 Amortization and simple interest

Amortization recurrence

Let $B_0 = \text{PV}$, $P = \text{PMT}$, $r = i/100$, and let n_0 be the stored count of elapsed payments. For a requested block of k payments, the recurrence is

$$\begin{aligned} I_j &= \begin{cases} 0 & \text{BEGIN and } n_0 + j - 1 = 0, \\ -rB_{j-1} & \text{otherwise,} \end{cases} \\ D_j &= P - I_j, \\ B_j &= B_{j-1} + D_j, \qquad j = 1, \ldots, k. \end{aligned}$$

Here I_j and D_j are signed interest and principal portions from the same viewpoint as the payment. AMORT returns their block totals, updates PV to B_k, and updates n to $n_0 + k$. FV does not participate. See section 11.11 for result placement.

The input PV and PMT must be nonzero and opposite in sign; i must exceed -100; n must be a nonnegative integer. In RPN, all four registers must actually be present, rather than merely supplying unwritten zeros. ALG validates the corresponding accepted worksheet inputs. The block count is an integer from zero through 100,000. A zero block still validates the financial setup. There is no automatic stop at an original loan term or at payoff, and no per-payment currency rounding. Consequently, this recurrence need not reproduce a lender's statement that rounds every posting to cents.

Simple interest

For simple interest, let $P = -\text{PV}$, $d = n$ be the day count, and $j = i/100$ be the annual rate fraction. The returned interest amounts are

$$I_{360} = Pj\frac{d}{360}, \qquad I_{365} = Pj\frac{d}{365}.$$

The principal P is also returned as a separate result. All three inputs must be finite; RPN additionally requires the three registers to have been stored. The day count $d \geq 0$ may be fractional. This command permits any finite annual rate, including values outside the TVM rate domain. BEGIN/END and PMT/FV do not enter the formulas. See section 13.2.

C.8 Cash-flow discounting and internal rate of return

For the selected cash-flow prefix $C_0, C_1, \ldots, C_N$,

$$\text{NPV}(i) = \sum_{j=0}^{N} \frac{C_j}{(1 + i/100)^j}. \tag{C.6}$$

Each position is one equally spaced period; position zero is now. The n input is the final included index, not a count of payments or a frequency. It must be an integer from 0 through 999 within the available cash-flow list. NPV requires finite $i > -100$. RPN reads explicitly stored n and i; ALG uses the corresponding accepted worksheet inputs. The computation uses reverse Horner evaluation: start with zero, then successively replace V by $C_j + V/q$, from the last selected flow back to the first, where $q = 1 + i/100$.

IRR finds a rate for which $\mathrm{NPV}(i) = 0$. At least two selected flows are required, with both positive and negative values. An all-zero stream is indeterminate. The selected prefix is normalized by its largest absolute amount; zero flows at its ends can be removed for the search without changing any finite root. In the transformed variable $t = \ln(1 + i/100)$, the residual is evaluated with decay factors $e^{-|t|}$ and a direction chosen to avoid unnecessary exponential growth. This rescales the equation without changing its finite zeros.

Without explicit bounds, RPN uses a stored valid i as the guess, or zero when it is unwritten; ALG uses its accepted initial-guess field. Optional bounds must be ordered, exceed -100, and either contain a sign change or give a computed zero at an endpoint. A stream with several sign changes can have several IRRs; the root found depends on the guess and bracket. An even-multiplicity root may not produce a sign change at all. A failed bracket search is not a proof that no mathematical root exists. The controls are in section 12.8; numerical stopping is described in section C.9.

NPV does not store PV, and IRR does not store i. BEGIN/END has no effect on these equally spaced cash-flow positions. The bank holds at most 1,000 expanded flows including C_0. A repetition count f from 1 through 1,000 appends $f - 1$ additional copies at the actual end of the list, subject to remaining capacity. Repetition is expansion, not a compressed weighting in equation (C.6).

C.9 Financial numerical root finding

TVM interest, IRR, and multi-coupon bond yield use bracketed numerical searches. Rates are transformed to $t = \ln(1 + r)$, which automatically keeps $r > -1$. For a search without supplied bounds, the first bracket step is 0.01 in t; the step doubles during expansion, testing the lower side first. TVM begins at $t = 0$, IRR uses its rate guess, and the bond yield calculation starts from a 3.5 percent semiannual rate, equivalent to 7 percent nominal annual yield.

Within a valid bracket, the solver alternates midpoint and secant candidates, using only candidates inside the bracket. Its standard width criterion is

$$|b - a| \le 10^{-30} \max(1, |a|, |b|), \tag{C.7}$$

where a, b are bounds in t, not percentages. A computed zero residual also stops the search, as does a candidate indistinguishable from an endpoint at the available precision. There are 200 iterations per search stage and a 100,000-evaluation budget. These settings are not user controls.

The tolerance is not a promise of 30 correct decimal places in the displayed rate. Transforming back to a periodic percentage multiplies small errors in t by approximately $100e^t$; the bond's nominal annual percentage has a factor $200e^t$. Finite-precision residuals, cancellation, conditioning, multiple roots, and the precision of the input values also matter. A computed zero is a zero of the evaluated residual, not a proof that the exact residual vanishes. Increasing working precision improves arithmetic but does not change this fixed search tolerance.

C.10 Gregorian dates and day counts

Encoding and validation

The accepted calendar is proleptic Gregorian, with years 1 through 9999. A leap year is divisible by 4 except that a year divisible by 100 must also be divisible by 400. Numeric date encodings are

$$\text{MDY}: \ M + \frac{D}{100} + \frac{Y}{10^6}, \qquad \text{DMY}: \ D + \frac{M}{100} + \frac{Y}{10^6}.$$

Thus the final four decimal places are the four-digit year. Display rounding does not add or remove stored date fields.

The decoder scales by 10^6 using at least 64 bits and at least the current and input precisions, rounds to an integer, validates the calendar fields, then re-encodes the date at the input precision. It requires equality with that canonical encoding. Extra nonzero date digits are therefore not silently treated as a fractional day. See section 13.3 for safe date entry.

Actual days and the implemented 30/360 rule

Let $Y' = Y - 1$. An ordinal with 1 January of year 1 equal to zero is

$$O(Y, M, D) = 365Y' + \left\lfloor \frac{Y'}{4} \right\rfloor - \left\lfloor \frac{Y'}{100} \right\rfloor + \left\lfloor \frac{Y'}{400} \right\rfloor + \sum_{m=1}^{M-1} \text{days}(Y, m) + D - 1.$$

Actual days from date 1 to date 2 are $O_2 - O_1$: the calculation is an elapsed interval, not an inclusive count of both endpoints. Date addition requires an integer offset between −3,652,058 and 3,652,058, and the resulting date must remain in years 1–9999.

For the second day-count result, form

$$d_1 = \begin{cases} 30 & D_1 = 31 \text{ and } D_2 \geq 30, \\ D_1 & \text{otherwise,} \end{cases} \qquad d_2 = \min(D_2, 30).$$

Then the implemented 30/360 count is

$$360(Y_2 - Y_1) + 30(M_2 - M_1) + d_2 - d_1. \tag{C.8}$$

There is no special February-end adjustment. This exact rule matters: it gives 27 from 31 January to 28 February in the same year, but −28 in the reverse direction. Do not assume antisymmetry or substitute a different market convention merely because both are called 30/360.

C.11 Semiannual bonds

The bond functions model redemption at 100, a nonnegative annual coupon percentage stored in PMT, and semiannual coupon payments. Put

$$C = \text{PMT}/2, \qquad r = i/200, \qquad q = 1 + r.$$

Prices and coupon amounts are per 100 of redemption value. Settlement must precede maturity. Coupon dates are obtained by moving backward in six-month steps from maturity. A maturity on a month's last day preserves month-end; otherwise a date uses the maturity's day number, clipped when necessary to the target month's last day. A preceding coupon date must exist within the supported calendar. See section 13.6.

Let N be the number of coupons strictly after settlement. Let E be the actual days in the coupon period containing settlement, and d the actual days from settlement to its next coupon. Define

$$w = d/E, \qquad 0 < w \leq 1, \qquad A = C(1 - w),$$

where A is accrued interest. Settlement on a coupon date excludes that day's coupon, uses the following coupon as the next one, and gives $w = 1$, $A = 0$.

With at least two remaining coupons, the dirty price and clean price are

$$P_{\text{dirty}} = \sum_{j=0}^{N-1} \frac{C}{q^{w+j}} + \frac{100}{q^{w+N-1}}, \qquad P_{\text{clean}} = P_{\text{dirty}} - A. \tag{C.9}$$

The implementation evaluates the coupon series in a stable nested form and uses logarithmic growth for the fractional exponent. With only one remaining coupon, it uses simple interest over the remaining fraction of a period:

$$P_{\text{dirty}} = \frac{100 + C}{1 + rw}, \qquad P_{\text{clean}} = P_{\text{dirty}} - A. \tag{C.10}$$

This final-period formula is deliberately different from substituting $N = 1$ into the compound formula.

Price requires a finite yield with $i > -200$, corresponding to $r > -1$, and a finite nonnegative coupon. Yield requires a finite positive clean price in PV and a finite nonnegative coupon. RPN requires these financial registers to have been stored; ALG uses the accepted worksheet inputs. For one remaining coupon, its direct nominal annual percentage is

$$i = \frac{200}{w}\left(\frac{100 + C}{\mathrm{PV} + A} - 1\right),$$

subject to the same rate domain. For several coupons, numerical root finding solves the clean-price equation; section C.9 describes the search.

PRC stores the calculated clean price in PV; YTM stores the nominal annual yield percentage in i. The stored n, FV, and BEGIN/END do not participate. These functions do not model odd coupon periods, a call date, alternative redemption amounts, or another coupon frequency. See section 13.7.

C.12 Depreciation methods

Let $K = \mathrm{PV}$ be cost, $S = \mathrm{FV}$ salvage value, $L = n$ life in years, and $B = K - S$ the depreciable basis. Inputs require finite values with $K \geq S \geq 0$, $L > 0$, and a requested integer year $j \geq 1$. PMT does not participate. Each method returns the depreciation for year j and the depreciable amount remaining *after* that year, excluding salvage. RPN requires the input registers to have been stored; ALG uses its accepted worksheet inputs. The depreciation calculation itself does not rewrite the financial input registers.

Set $m = \lceil L \rceil$. A requested year beyond m, or a zero basis, gives zero for both results. For the declining-balance methods, the factor is still validated before that shortcut. Fractional lives are accepted, with the last active year handled as specified below; no automatic currency rounding is performed.

Straight line

The fraction of a full year still available in year j is $u_j = \min(1, L - j + 1)$. Within the active life,

$$D_j = \frac{B}{L}u_j, \qquad R_j = \max\left(0, B - \frac{B}{L}\min(j, L)\right).$$

Thus a final fractional year receives the corresponding fraction of a full-year charge.

Sum of the years' digits

The implemented weights for a possibly fractional life are $L, L-1, \ldots, L-m+1$. Their sum and the cumulative first j weights are

$$W = mL - \frac{m(m-1)}{2}, \qquad U_j = jL - \frac{j(j-1)}{2}.$$

For $1 \le j \le m$,

$$D_j = B\frac{L-j+1}{W}, \qquad R_j = \max\left(0, B - B\frac{U_j}{W}\right).$$

For integer L, this becomes the familiar sum-of-digits denominator $L(L+1)/2$. For fractional L, the stated weights, rather than an assumed integer-life formula, determine the result.

Declining balance and the straight-line switch

Let the positive factor be $F = i$ percent and set $q = (F/100)/L$. Start with book value $V_0 = K$. In each active year h, define

$$B_h = \max(0, V_{h-1} - S), \qquad L_h = L - h + 1.$$

For ordinary declining balance,

$$D_h = \begin{cases} B_h & L_h \le 1, \\ \min(B_h, qV_{h-1}) & L_h > 1. \end{cases}$$

For declining balance with the straight-line switch,

$$D_h = \begin{cases} B_h & L_h \le 1, \\ \min(B_h, \max(qV_{h-1}, B_h/L_h)) & L_h > 1. \end{cases}$$

Then $V_h = V_{h-1} - D_h$, and the requested remaining basis is $R_j = \max(0, V_j - S)$. The last active year writes off the entire remaining basis. Earlier charges cannot push book value below salvage. The switch method compares the declining charge with straight-line depreciation of the remaining basis over the remaining life on every iteration.

These two iterative methods permit at most 100,000 active years in a requested calculation. A request beyond the asset's life, or for a zero basis, returns via the earlier zero-result rule without iterating through those years. See chapter 13 for the register setup and commands.

C.13 Conversion definitions and factor catalog

Scale, offset, and evaluation

Each ordinary unit has a scale a/b and offset o relative to a category's base unit. The conversion from source to destination is

$$v_{\text{base}} = (v + o_s)\frac{a_s}{b_s}, \qquad v_d = v_{\text{base}}\frac{b_d}{a_d} - o_d. \tag{C.11}$$

All offsets are zero except the temperature offsets listed below. Operations are performed in the shown add–multiply–divide order at current working precision; mathematically canceling factors are not a promise of a bitwise reversible round trip. A same-unit conversion also produces a value at the current precision. Conversions use the real component; they do not convert both components of a complex pair.

Every terminating decimal and rational factor in table C.2 is the exact *catalog definition*. This statement distinguishes the application's chosen constants from the rounded binary values used in arithmetic; it does not assert that all physical realizations or similarly named customary units are identical. The catalog distinguishes U.S. and Imperial liquid measures, thermochemical calories, International Table BTU, and mechanical horsepower. Angle conversion additionally depends on an approximation to π. See section 9.9.

Table C.2: Complete multiplicative catalog. Factors give base units per named unit. Units sharing a row have factors in the same order.

Unit	Factor
Length: base metre (m)	
m; km; cm; mm	1; 1000; 10^{-2}; 10^{-3}
micrometre; nanometre	10^{-6}; 10^{-9}
inch; foot; yard	0.0254; 0.3048; 0.9144
mile; nautical mile	1609.344; 1852
Area: base square metre (m^2)	
m^2; km^2; cm^2; mm^2	1; 10^6; 10^{-4}; 10^{-6}
square inch; square foot; square yard	0.00064516; 0.09290304; 0.83612736
acre; hectare	4046.8564224; 10000
Volume: base litre (L)	
L; mL; cubic metre; cubic centimetre	1; 0.001; 1000; 0.001
cubic inch; cubic foot	0.016387064; 28.316846592
U.S. fluid ounce	0.0295735295625
U.S. pint; quart	0.473176473; 0.946352946
U.S. gallon	3.785411784
Imperial fluid ounce	0.0284130625
Imperial pint; quart; gallon	0.56826125; 1.1365225; 4.54609

Continued on next page

Unit	Factor
Mass: base kilogram (kg)	
kg; gram; milligram; tonne	1; 0.001; 10^{-6}; 1000
ounce; pound	0.028349523125; 0.45359237
stone; short ton; long ton	6.35029318; 907.18474; 1016.0469088
Speed: base metre per second (m/s)	
m/s; km/h	1; 5/18
mile per hour; foot per second	1397/3125; 381/1250
knot	463/900
Acceleration: base $\mathrm{m/s^2}$	
$\mathrm{m/s^2}$; foot per second squared	1; 0.3048
standard gravity	9.80665
Force: base newton (N)	
N; kN; pound-force	1; 1000; 4.4482216152605
Pressure: base pascal (Pa)	
Pa; kPa; MPa	1; 1000; 10^6
bar; millibar; atmosphere	100000; 100; 101325
pound-force per square inch (psi)	8896443230521/1290320000
torr	101325/760
Energy: base joule (J)	
J; kJ; MJ	1; 1000; 10^6
watt-hour; kilowatt-hour	3600; 3600000
thermochemical calorie; kilocalorie	4.184; 4184
International Table BTU	1055.05585262
Power: base watt (W)	
W; kW; MW	1; 1000; 10^6
mechanical horsepower	745.69987158227022
Time: base second (s)	
s; millisecond; microsecond; nanosecond	1; 10^{-3}; 10^{-6}; 10^{-9}
minute; hour; day	60; 3600; 86400
Frequency: base hertz (Hz)	
Hz; kHz; MHz; GHz	1; 1000; 10^6; 10^9
Torque: base newton metre (N m)	
newton metre; pound-force foot	1; 1.3558179483314004

Table C.3: Temperature definitions relative to kelvin, using $v_K = (v + o)a/b$.

Unit	a	b	o
kelvin	1	1	0
degree Celsius	1	1	273.15
degree Fahrenheit	5	9	459.67
degree Rankine	5	9	0

Temperature conversion is affine rather than multiplicative. The converter applies these formulas without enforcing an absolute-zero physical limit. A temperature difference is a different quantity: applying the absolute temperature offset to a difference would be inappropriate.

The angle catalog is radian, degree, gradian, and turn, with radian factors

$$1, \qquad \pi/180, \qquad \pi/200, \qquad 2\pi$$

respectively. Angle conversion explicitly passes through radians using the current value of π. Selecting source and destination angle units in the converter does not change the calculator's DEG/RAD/GRAD setting.

C.14 Limits and independent checks

Table C.4: Principal numerical and operational limits.

Quantity	Current limit or rule
Working precision	16–10,000 decimal digits through settings; default 600.
Display digits	0 through the current working digit count; default FIX 4.
RPN stack setting	Fixed depth 2–1,000, or 0 for unlimited; default 5. Unlimited remains constrained by available resources.
Ordinary memory	Ten registers, R0–R9.
Real-mode factorial	Integer argument from 0 through 10,000. Complex mode also permits the Gamma extension, including noninteger real inputs; see section C.4.
Permutations and combinations	Integer real components with $0 \le k \le n \le 10{,}000$, even in complex mode.
Real integer root index	Nonzero, between −1,000,000 and 1,000,000.
Complex powers	MPC power and integer-exponent primitives; finite exponent range and working precision still apply.
Complex gamma	Requested precision no greater than 10,000 decimal digits; poles excluded.
Nonreal erf and erfc	Approximate radius-100 input limit and finite convergence budget; real arguments use a separate path. See section C.4.

Continued on next page

Quantity	Current limit or rule
Cash-flow bank	1,000 expanded positions including CF0; selected last index 0–999 within the list.
Cash-flow repetition	Integer frequency 1–1,000, subject to total bank capacity.
AMORT block	Integer count 0–100,000.
Iterated depreciation	At most 100,000 active years for DB or the DB/SL switch.
Dates	Gregorian years 1–9999; integer date-addition offsets within ±3,652,058, with the result still in range.
Financial root search	200 iterations per stage, 100,000 evaluations, fixed transformed-variable tolerance; see section C.9.
ALG streaming editor	4,096 tokens, nesting depth 128, and a 10,000-character numeric-entry stream limit.
ALG worksheet field	100,000 source characters, 4,096 tokens, and nesting depth 128.

The RPN numeric-entry editor has no configured character cap. Available memory and processing time still place practical limits on long entries and unlimited stacks.

An effective numerical check matches the problem's structure. Substitute a TVM result back into equation (C.5); evaluate NPV at an IRR; compare book-value changes with depreciation; check a conversion's units and inverse; and compare a regression with centered data. Use freshly entered inputs when changing precision, preserve cash signs and period units, and distinguish a different branch or date convention from a numerical error. The worked chapters supply complete operating examples; chapter 14 explains how to diagnose unexpected behavior and preserve a reproducible calculation.

Appendix D

Obtaining and Building the Source

SpicyRPN is an Xcode project for native macOS, iPhone, and iPad applications. It supports iOS/iPadOS 18.0 and later, and macOS 15.0 Sequoia and later on compatible Apple-silicon and Intel Macs. The Mac application is universal, with arm64 and x86_64 code. The project links three static libraries, GMP, MPFR, and MPC, which must be built for the destination before building the app. GMP and MPFR provide the real-number foundation; MPC provides complex arithmetic and elementary functions for both entry modes. SQLite is compiled from source as part of the app. The manual has a separate LaTeX build and does not require Xcode.

This appendix describes the Release 2.0.0 source package and scripts. Retain the package version, its published checksum, and any local changes when building your own copy. Deployment settings below are requirements in the project, not a record of successful tests on each listed system.

D.1 Preparing the development machine

Use a Mac supported by the selected full Xcode installation, with the platform components needed for your destination. The macOS library builder accepts native arm64 and x86_64 hosts; the installed toolchain must support both target architectures to build the universal libraries. The project was created with Xcode 27-era tools. Although its Swift language setting is Swift 5, that setting is not a statement that an older Swift 5 toolchain can build the project. Check Apple's current Xcode requirements and SDK table when choosing an installation: `https://developer.apple.com/xcode/system-requirements`.

Launch Xcode once to complete its setup and install the required platforms. Select that Xcode installation in the Command Line Tools setting. A standalone Command Line Tools installation is not a substitute for the full Xcode application when building and testing SpicyRPN. Use a native Terminal session: arm64 on Apple silicon or x86_64 on Intel. If more than one Xcode is installed, select the intended installation before checking the SDKs. A shell-local selection does not change the machine-wide setting. Adjust this path to your installation before running the export:

```
export DEVELOPER_DIR=/Applications/Xcode.app/Contents/Developer
```

These commands show the active environment:

```
uname -m
xcode-select --print-path
xcodebuild -version
xcodebuild -showsdks
xcrun --sdk macosx --show-sdk-path
```

The host architecture should be `arm64` on Apple silicon or `x86_64` on Intel. The selected developer path, or the `DEVELOPER_DIR` override when set, should belong to the intended Xcode installation. Rosetta is optional for separately testing the Intel application on Apple silicon; it is not required to compile that slice.

Apple's description of this selection mechanism is at `https://developer.apple.com/documentation/xcode/configuring-command-line-tools-settings`.

Table D.1: Deployment targets in the Release 2.0.0 Xcode project.

Target	iOS	macOS
`SpicyRPN`	18.0	15.0
`SpicyRPNTests`	18.0	15.0
`SpicyRPNUITests`	18.0	15.0

These values apply to both Debug and Release configurations and match the math-library defaults. Choose a simulator or device meeting these minimums. The Xcode installation may require a newer development Mac operating system than the app requires at runtime. The app supports iPhone and iPad device families and native macOS; Mac Catalyst is disabled. Other platform settings appearing in test project metadata do not establish an additional supported app platform.

The macOS recipe builds both arm64 and x86_64 libraries and merges them into universal archives. The iOS device and simulator recipes remain arm64-only; Intel simulator libraries are not supplied. An arm64 iOS device archive and an arm64 iOS simulator archive are different platform binaries, and neither can substitute for a macOS archive.

The scripts use Bash, Make, the Xcode C compiler and SDK tools, and standard macOS utilities including `curl`, `tar`, `shasum`, and `sysctl`. Keep the extracted source in a path without spaces, as required by the upstream Autoconf builds. The Xcode and SDK paths must also have no spaces; the library builders reject them otherwise. Install Python 3.9 or later before bundle verification. The optional reference-fixture workflow in section D.12 requires Python 3.11 or later. Install full MacTeX, including LuaLaTeX, `latexmk`, and `makeindex`, before building the manual; see section D.13 for the build command.

D.2 Obtaining the release source package

Download `SpicyRPN-2.0.0-source.zip` and its checksum file from `https://spicyrpn.com/`. The same site supplies the matching `SpicyRPN-2.0.0-Manual.pdf` and a ZIP containing that PDF. Keep an unchanged copy of the source ZIP. Work in its download directory, whose parent path must have no spaces for the subsequent library builds:

```
spicy_archive=SpicyRPN-2.0.0-source.zip
```

Compute the source archive's digest:

```
shasum -a 256 "$spicy_archive"
```

Compare the result with the archive's value in `SHA256SUMS.txt`; do not extract a mismatched file. Once it matches, use a fresh validation directory. The next command removes only the previous generated `build/source-check/` copy; keep any changes you need elsewhere first. Then extract the ZIP and verify its contents:

```
rm -rf build/source-check
mkdir -p build/source-check
unzip "$spicy_archive" -d build/source-check
cd build/source-check/SpicyRPN-2.0.0
shasum -a 256 -c SOURCE-SHA256SUMS.txt
```

Proceed only after every source checksum passes. All app and test commands below run from this extracted `SpicyRPN-2.0.0` source root. Quote paths containing spaces, including the manual and SQLite subdirectories. For a release validation, keep this source set unchanged; if a correction is needed, fix the working tree, rebuild the package, and start again with its fresh extraction. Record personal-build modifications separately.

The complete package includes verified GMP, MPFR, and MPC source archives under `ThirdParty/`, SQLite sources under `SpicyRPN/Third Party/`, the Xcode project, app and test sources, build scripts, licensing notices, and editable manual. The math builders extract fresh working sources from the archives; the bundled MPC archive allows an offline library build after Xcode and its SDKs are installed. See section D.5 for the MPC source selection.

Read `LICENSE` and `LICENSING.md` alongside the source. The SpicyRPN application and original manual use MIT; bundled components retain their own licensing notices. Appendix E reproduces the complete MIT, GNU GPLv3, and LGPLv3 texts. Keep those notices and the corresponding source and build material when preparing a source distribution. Signing a personal build does not change its licensing terms.

D.3 Understanding the source layout

Table D.2: Files and directories used when building SpicyRPN.

Location	Purpose
`SpicyRPN.xcodeproj/`	App, unit-test, and UI-test targets; the shared `SpicyRPN` scheme.
`SpicyRPN/Preview/`	Calculator model, presentation, and interface implementation used by the app.
`SpicyRPN/Math/`	Multiprecision values, precision handling, and complex arithmetic.
`SpicyRPN/MathBridge/`	C bridge to GMP, MPFR, and MPC.
`ThirdParty/`	GMP 6.3.0, MPFR 4.2.2, and MPC 1.4.1 source archives with their upstream notices.
`build/downloads/`	Optional verified MPC archive cache, used when the archive is absent from `ThirdParty/`.
`ThirdParty/SHA256SUMS.txt`	Checksums for the three bundled source archives.
`SpicyRPN/Third Party/`	SQLite 3.51.2 amalgamation and associated headers.
`Scripts/`	Platform-specific math builds and independent test-reference generators.
`SpicyRPNTests/`	Native model, arithmetic, and other unit tests.
`SpicyRPNUITests/`	XCTest tests that operate the launched calculator through its interface.
`Docs/`	Project notes, including test architecture and coverage documentation.
`SpicyRPN Manual/`	Manual sources and PDF build scripts.
`build/install/`	Generated GMP, MPFR, and MPC installations, separated by destination.

The project links the static libraries produced by its own scripts; a package manager's GMP, MPFR, or MPC installation does not replace that step. There is no Swift package, CocoaPods, or Carthage dependency installation step in this build. SQLite's C source is compiled by Xcode and does not need a separate library-build command.

To check the bundled source archives independently of a build, run:

```
(
  cd ThirdParty
  shasum -a 256 -c SHA256SUMS.txt
)
```

The scripts also check the expected archive hashes and extract their working sources from those archives. Expanded reference trees, when present in a working copy, do not enter the build automatically. If an archive fails verification, obtain the correct file from the matching release source package before continuing.

D.4 Library deployment settings

All three libraries share an explicit minimum for their destination:

```
MACOSX_DEPLOYMENT_TARGET=15.0
IOS_SIM_DEPLOYMENT_TARGET=18.0
IOS_DEVICE_DEPLOYMENT_TARGET=18.0
```

These defaults match the app and test targets. If a Terminal session has older overrides, clear them before building for the supported baseline:

```
unset MACOSX_DEPLOYMENT_TARGET
unset IOS_SIM_DEPLOYMENT_TARGET IOS_DEVICE_DEPLOYMENT_TARGET
```

These environment variables can deliberately override library compiler targets; they do not change the app or test targets in Xcode. Keep the settings aligned and rebuild the libraries and app after changing them. A library built for an earlier minimum does not lower the app's minimum, and a library requiring a later system cannot establish support for the app's earlier baseline.

D.5 Building GMP, MPFR, and MPC

Every library build replaces its generated sources and working trees. Preserve dependency patches separately before running a builder; see section D.9. The deployment settings above must match your intended baseline. Choose only the destinations you need; these are three separate commands, not three mandatory stages of every build:

```
bash Scripts/build-math-macos-universal.sh
bash Scripts/build-math-ios-simulator-arm64.sh
bash Scripts/build-math-ios-device-arm64.sh
```

Each script builds GMP 6.3.0 first, MPFR 4.2.2 against that GMP installation, and then MPC 1.4.1 against both. The macOS, iOS simulator, and iOS device outputs use suffixes `macos-universal`, `ios-simulator-arm64`, and `ios-device-arm64`, respectively. For example, the macOS installations are:

```
build/install/gmp-macos-universal/
build/install/mpfr-macos-universal/
build/install/mpc-macos-universal/
```

The macOS command builds each architecture separately, then merges both static-library slices. The thin-architecture scripts `Scripts/build-math-macos-arm64.sh` and `Scripts/build-math-macos-x86_64.sh` are available for library development. Their `macos-arm64` and `macos-x86_64` installations do not replace the universal prefixes selected by Xcode. Run the universal script after a library change to refresh both slices.

Each prefix contains its generated headers and static library. Xcode's SDK-specific header and library search paths select the matching prefixes; both Debug and Release link using `-lmpc -lmpfr -lgmp`, in dependency order. There is no Xcode build phase that automatically runs these scripts when a library is missing.

Table D.3: Checks performed by the math-library build scripts.

Destination	Build and verification behavior
macOS	Builds arm64 and x86_64, checks each slice's architecture, platform, and deployment target, and links the math bridge and MPC inverse-function smoke program for each. The native host slice runs upstream `make check` and the MPC program; the cross slice is checked by compilation and linking. Both are merged into universal static libraries.
iOS simulator	Cross-compiles arm64 simulator libraries, with GMP assembly disabled. Reports archive architecture and sample object platform metadata. Links the actual math bridge and the MPC inverse-function smoke program; does not run that program or upstream test executables.
iOS device	Cross-compiles arm64 device libraries, with GMP assembly disabled. Checks arm64 architecture and sample object iOS platform metadata, then links the actual math bridge against the libraries. Also links the MPC inverse-function smoke program. These checks establish linking, not execution on a device.

The entry scripts share `Scripts/lib/build-math.sh`. Both `--clean` and `-clean` remove the selected destination's generated work and installations, retaining archives and other destinations. For universal macOS this includes both thin builds and the merged installation. Keep any patches outside those directories before a clean rebuild. Run only the destinations you need:

```
bash Scripts/build-math-macos-universal.sh --clean
bash Scripts/build-math-ios-simulator-arm64.sh --clean
bash Scripts/build-math-ios-device-arm64.sh --clean
```

Use `--help` to display the options. Xcode's Clean Build Folder is a separate app-build operation.

The scripts verify minimum-OS metadata as well as platform metadata in a sample object from each archive and in the linked bridge and MPC smoke program. Each must match the selected deployment target. The check reads the operating-system deployment field from its Mach-O load command; embedded linker and other tool versions do not describe the minimum OS.

The macOS builds retain GMP's optimized assembly for each architecture. Its `CCAS` command includes `-c` so that GMP's assembly probes and build rules produce object files without attempting to link an executable. The iOS builds disable GMP assembly.

MPC source and integration

MPC provides arbitrary-precision complex arithmetic on top of MPFR and GMP. The calculator calls it through a C bridge shared by RPN and ALG. The library for the selected destination is required to build the app. See chapter 8 and section C.4 for numerical conventions and the remaining limits of composed and special-function calculations.

MPC is part of each unified build. The release source package includes `ThirdParty/mpc-1.4.1.tar.xz`. Its upstream archive is `https://ftp.gnu.org/gnu/mpc/mpc-1.4.1.tar.xz`. The builder prefers the bundled archive; if it is absent, the builder reuses `build/downloads/mpc-1.4.1.tar.xz` or downloads it there. Every path is checked against the pinned SHA-256 before use:

```
91204cd32f164bd3b7c992d4a6a8ce6519511aadab30f78b6982d0bf8d73e931
```

A complete release ZIP therefore supplies MPC without a network request. A checksum mismatch stops the build. Retain the verified archive, its licensing notices, any local patches, and build settings with source distributions. MPC uses LGPL-3.0-or-later; see `LICENSING.md`.

The builder selects the SDK, compiler, architecture, static-library configuration, and matching dependency prefixes. It installs `include/mpc.h` and `lib/libmpc.a` beneath `build/install/mpc-<target>/`. The generated source and build trees are replaced on every invocation, so preserve patches separately.

After the libraries are built, follow the signing and application-build sections below. Header and library search paths and linker flags are already configured; no manual

library attachment or Xcode build phase is needed. The macOS upstream tests and smoke program check MPC itself. Mobile smoke checks only compile and link; native app tests remain a separate step.

The C bridge transfers `CalcNumber`'s MPFR components directly, preserving its value representation and calculation provenance. Complex arithmetic and elementary functions use MPC, while valid real shortcuts retain MPFR behavior. Explicit signed-zero choices preserve the documented axis branches. Angle scaling and custom Gamma and error-function algorithms use appropriate intermediate precision and retain their own accuracy limits. The detailed setup and runtime description is in `Scripts/BUILDING.md`.

D.6 Signing and personal runs

For a personal build, configure signing before any application build, run, test, or archive. For a release, settle these project settings in the working tree before packaging, then verify them without changing the extracted sources. Open the project with:

```
open SpicyRPN.xcodeproj
```

The checked-in project uses automatic signing and contains the maintainer's team and bundle identifiers. For your own build, add your Apple Account in Xcode's settings, then select your team in the app target's Signing & Capabilities pane. Use a bundle identifier that is valid and available for your team. If Xcode offers a Personal Team, it can be used within Apple's current personal-development limits. Do not assume access to the maintainer's signing identity.

Check the unit-test and UI-test targets as well when preparing to run tests. Their signing configuration must work with your account and the selected destination. Native macOS development on your own Mac does not generally require registering that Mac as an iOS-style provisioned device, but project signing settings and enabled capabilities still need to be compatible with your local setup.

For a development-signed iPhone or iPad app, follow Xcode's device setup prompts. Developer Mode is managed in the device's Settings under Privacy & Security; enabling it involves a restart and confirmation. The setting may become available after connecting the device to Xcode. Apple maintains the detailed procedures here:

- Running your app on simulated or physical devices.
- Enabling Developer Mode on a device.

A different app identity can have a different sandbox and saved calculator state. Do not expect a personal build with a changed bundle identifier to inherit another installation's database.

For disposable manual experiments, add `SPICYRPN_EPHEMERAL=1` to the Run scheme's environment. This makes the calculator use an in-memory database for that run. Remove the variable to return to ordinary persistent storage; using the variable does not delete an existing on-disk database. For normal state and recovery behavior, see chapter 14.

D.7 Building and running on macOS

With the macOS libraries built and signing configured, select the shared `SpicyRPN` scheme and a native My Mac run destination. Configure signing as described in section D.6, then use Product → Build or Product → Run. The normal application launches the calculator interface. A foundation diagnostic view available as a SwiftUI preview is not the app's main screen.

For command-line work, inspect the actual schemes and destinations:

```
xcodebuild -list -project SpicyRPN.xcodeproj
xcodebuild -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -showdestinations
```

Build a Debug app for the native host architecture with:

```
spicy_mac_arch="$(uname -m)"
xcodebuild build -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination "platform=macOS,arch=$spicy_mac_arch" \
  -derivedDataPath "build/DerivedData-mac-$spicy_mac_arch"
```

This command builds the product; it does not launch it. Xcode's Run command handles launching and attaching its debugger. The chosen derived data directory holds Xcode products and intermediates. It is separate from `build/install/`, which holds the math libraries.

D.8 Building for iPhone and iPad

Simulator

Build the arm64 simulator math libraries on an Apple-silicon Mac, install a suitable iOS simulator runtime in Xcode, and select an arm64 iPhone or iPad simulator as the run destination. Both the app and tests require an iOS 18.0 or later runtime. A simulator's device name alone does not identify its installed runtime.

Obtain the simulator identifier from `xcodebuild -showdestinations` with the project and scheme options shown in section D.7. Then replace the placeholder:

```
spicy_simulator='REPLACE_WITH_SIMULATOR_UDID'
xcodebuild build -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination "platform=iOS Simulator,id=$spicy_simulator" \
  -derivedDataPath build/DerivedData-simulator
```

Use Xcode's Run command to launch the app on the selected simulator. The simulator build uses its own static libraries even on an arm64 Mac; it cannot use the macOS or physical-device library installations.

Physical device

Build the device math libraries, connect and unlock the iPhone or iPad, and complete any trust or pairing steps shown by Xcode and the device. Configure your signing team and enable Developer Mode where required, as described in section D.6. Choose the actual device and use Run to build, install, and launch the app.

For a command-line build, use a device identifier reported as an available destination:

```
spicy_device='REPLACE_WITH_DEVICE_UDID'
xcodebuild build -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination "platform=iOS,id=$spicy_device" \
  -derivedDataPath build/DerivedData-device
```

This build command does not itself install or launch the product. A generic iOS build destination can produce build artifacts, but it is not a runnable test destination. Keep the simulator and device derived data directories distinct when using the recipes above.

D.9 Rebuilding after modifying a library

The release package supplies the application and library sources for the LGPLv3 section 4(d)(0) rebuilding route. Use a separate extraction for a modified personal build. Prepare signing as described in section D.6 before running the resulting application.

Keep changes as unified-diff `.patch` files whose paths have one leading directory, such as `a/src/file.c` and `b/src/file.c`. The builder applies them with `patch -p1` relative to the library's source root. Create patch directories first:

```
mkdir -p build/library-patches/gmp \
  build/library-patches/mpfr build/library-patches/mpc
```

Save patches in the appropriate library subdirectory. Number multiple patches in application order, for example `01-change.patch` and `02-followup.patch`, then enable the patch directory:

```
export SPICYRPN_MATH_PATCH_DIR="$PWD/build/library-patches"
```

Every build verifies and extracts the original archives, then applies the patches in C-locale filename order before configuring any library. A failed patch stops the build. Both Mac architecture slices receive the same patches. The original archives and checksums stay unchanged; retain the patches separately. A library with no patches builds unchanged. Editing an expanded reference tree under `ThirdParty/` alone does not change the build.

Rebuild every destination affected by the change. The supplied recipes build all three libraries in the correct order, so a changed GMP is followed by MPFR and MPC linked against the matching installation. Use `--clean` to remove older generated working directories as well. Keep local patches outside those directories.

Then rebuild the app so that it links the new static libraries. For example, on macOS:

```
bash Scripts/build-math-macos-universal.sh --clean
spicy_mac_arch="$(uname -m)"
xcodebuild clean build -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination "platform=macOS,arch=$spicy_mac_arch" \
  -derivedDataPath "build/DerivedData-mac-$spicy_mac_arch"
```

Rebuilding a library does not update an already running app. Conversely, Xcode's Clean Build Folder command does not rebuild the libraries under `build/install/`. Treat the library and app rebuilds as separate steps.

For iPhone or iPad, use the device or Simulator library recipe for the selected destination while the patch setting is active, then clean and run the app from Xcode. Choose your own team and bundle identifiers as described in section D.6. Run the native tests against the rebuilt application.

To return to the unmodified libraries, run `unset SPICYRPN_MATH_PATCH_DIR`, rebuild the selected libraries, and clean/relink the app again. Keep the patches and build settings with any modified distribution. `Scripts/RELEASING.md` describes a separate modified-library rebuild check before publishing the release.

Replacing a dependency version requires reviewing the source archives, checksums, expanded directory names, build scripts, generated installation paths, project search paths, and bridge compatibility together. The scripts are pinned recipes, not automatic dependency upgraders. Retain the actual modified source and the build settings used for a modified binary, as well as the relevant licensing notices.

D.10 Running the native test suites

The shared scheme includes both `SpicyRPNTests` and `SpicyRPNUITests`. After preparing libraries and signing for the chosen destination, use Product → Test in Xcode. The unit-test target uses Swift Testing, including aggregate numerical test runners. The UI-test target uses XCTest to launch the actual app, press its controls, and read accessibility values. UI tests do not import the calculator's arithmetic implementation to manufacture expected answers.

The scheme permits unit tests to run in parallel and keeps the UI-test target serial. Preserve the serial UI configuration when diagnosing focus, modal, or input-order failures. Some unit-test runners control successful-case logging through a source-level verbosity constant; failures and their aggregate failure assertion remain active when that verbosity is disabled.

These are Apple-platform tests. Building documentation, checking scripts, or examining numerical fixtures on Linux does not establish that the app or its native test targets compile or pass in Xcode.

A complete command-line run

Run both test targets natively on an Apple-silicon Mac and on a compatible Intel Mac. The result-bundle path must not already exist. The following selects the current host architecture and creates a unique path for the results:

```
mkdir -p build/TestResults
spicy_mac_arch="$(uname -m)"
spicy_results="build/TestResults/$spicy_mac_arch-$(uuidgen)"
spicy_results="$spicy_results.xcresult"
xcodebuild test -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination "platform=macOS,arch=$spicy_mac_arch" \
  -derivedDataPath "build/DerivedData-mac-$spicy_mac_arch" \
  -resultBundlePath "$spicy_results"
open "$spicy_results"
```

For an iOS run, substitute an available simulator or device destination and its separate derived data path, and prepare the matching libraries first. The selected iOS destination must satisfy the test targets' iOS 18.0 minimum. A build, signing, or app-launch failure is not a completed test run.

The full UI suite includes long input sequences, large token-boundary cases, and broad conversion coverage. Allow a substantial run window; use a focused selection when investigating one failure. For example, this requests just the unit-test target for the native host:

```
spicy_mac_arch="$(uname -m)"
xcodebuild test -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination "platform=macOS,arch=$spicy_mac_arch" \
  -only-testing:SpicyRPNTests
```

To select UI tests instead, replace the final option with:

```
-only-testing:SpicyRPNUITests
```

A narrower selection can name a class:

```
-only-testing:SpicyRPNUITests/FinancialE2ETests
```

To select a current method within that class, use its identifier as reported by Xcode. Selecting tests controls what runs; Xcode may still build other targets required by the scheme.

Additional Intel testing with Rosetta

On Apple silicon with Rosetta installed, Xcode may offer an x86_64 Mac destination. Use `xcodebuild -showdestinations` to verify that it is available, then run a separate test pass:

```
xcodebuild test -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Debug \
  -destination 'platform=macOS,arch=x86_64' \
  -derivedDataPath build/DerivedData-mac-rosetta
```

This exercises the Intel slice under translation and is useful additional coverage. It does not replace a native run on a compatible Intel Mac. The library builder does not automatically execute cross-architecture tests through Rosetta. An Intel Mac cannot execute the arm64 slice; validate it on Apple silicon.

Reading a failure report

Open the result bundle or Xcode's Report navigator and start with the first failing test activity. Numerical UI checks retain full expected and observed strings and identify the first differing normalized decimal position. Read those details before reducing a failure to the short number visible in a display screenshot. For an input failure, establish whether the intended control was present, enabled, and hittable, and whether a sheet or keyboard focus intercepted the action.

Retain the source revision, Xcode version, destination and runtime, exact test identifier, and result bundle. Reproduce the smallest failing test, then run the related class or target after the change. Test behavior and coverage conventions are documented in `Docs/UI-Testing.md` and `Docs/UI-Test-Coverage.md`. Apple's test-selection and results workflow is described at `https://developer.apple.com/documentation/xcode/running-tests-and-interpreting-results`.

Database isolation in UI tests

The UI-test harness supplies a fresh UUID through `SPICYRPN_TEST_DATABASE`. The app then uses a test database under a UUID-specific temporary directory rather than its normal Application Support database. The harness clears calculator banks through the interface. Persistence tests can relaunch the app with the same UUID to check whether state survived.

Do not add `SPICYRPN_EPHEMERAL=1` to a persistence test run: in-memory storage takes precedence over the UUID database and changes what that test exercises. An arbitrary non-UUID value is not a valid isolated test-database identifier. Debug UI

tests can also supply a random seed, but the deterministic random source requires the UI-testing flag and a valid test UUID as well as that seed. These are harness controls, not calculator preferences.

D.11 Archiving and publishing the validated release

Before archiving, complete the native tests in section D.10 on Apple silicon and a compatible Intel Mac, and any iOS destinations being released. Keep the extracted sources unchanged and use the signing configuration prepared in section D.6. Verify the extracted source checksums again before archiving:

```
shasum -a 256 -c SOURCE-SHA256SUMS.txt
```

If a checksum no longer matches, resolve the source change and repackage before continuing.

The scheme uses Debug for running, testing, and analysis, and Release for profiling and archiving. Debug builds the selected active Mac architecture. Release builds include both arm64 and x86_64. Build a universal Mac archive and verify its executable with:

```
xcodebuild archive -project SpicyRPN.xcodeproj \
  -scheme SpicyRPN -configuration Release \
  -destination 'generic/platform=macOS' \
  -archivePath build/SpicyRPN-macOS.xcarchive \
  'ARCHS=arm64 x86_64' ONLY_ACTIVE_ARCH=NO
spicy_products=build/SpicyRPN-macOS.xcarchive/Products
spicy_binary="$spicy_products/Applications/SpicyRPN.app"
spicy_binary="$spicy_binary/Contents/MacOS/SpicyRPN"
xcrun lipo -archs "$spicy_binary"
xcrun lipo -verify_arch arm64 x86_64 "$spicy_binary"
```

The source package also supplies a bundle verifier. Pass the application bundle path, rather than its executable:

```
spicy_mac_app="$spicy_products/Applications/SpicyRPN.app"
python3 Scripts/verify-macos-universal.py "$spicy_mac_app"
```

It checks Release 2.0.0/build 1, the macOS deployment minimum, and both architectures in the main and bundled Mach-O files. The release packager can perform the same checks with its `--mac-app` option; see `Scripts/RELEASING.md`.

The architecture check verifies that both slices are present. It does not execute either slice or validate signing. Complete the App Store archive checks, including its signing, bundled privacy manifest, and configured license terms. Then publish the exact source ZIP, manual PDF, and checksum file whose sources you validated. These files remain in the original working tree's `dist/2.0.0/`; upload that set to `https://spicyrpn.com/`. If validation requires a source change, repackage and repeat the fresh-extraction checks before publication.

D.12 Checking independent numerical references

The included UI tests contain their expected values, so generating references is not a prerequisite for an ordinary app build or test run. The generator scripts are useful when reviewing a numerical change or adding cases. They use independent calculations rather than running SpicyRPN to obtain its own expected answers.

Use Python 3.11 or later for the documented generator workflow. The complex generator requires `mpmath` 1.3.0. The test documentation also records independent `mpmath` checks of selected scientific constants. An optional isolated environment can be prepared from the source root:

```
python3 -m venv build/fixture-env
source build/fixture-env/bin/activate
python -m pip install 'mpmath==1.3.0'
```

The scientific generator additionally needs an independently available MPFR shared library that Python can load. It discovers that library through the system library lookup unless `MPFR_LIBRARY` names an explicit library file. The static `libmpfr.a` produced for the app is not a loadable shared library. Configure the shared library for the Python process and host architecture on which the generator runs.

Run the comparison modes explicitly:

```
python Scripts/generate-scientific-ui-fixtures.py --check
python Scripts/generate-complex-ui-fixtures.py --check
python Scripts/generate-financial-ui-fixtures.py --check
python Scripts/generate-conversion-ui-fixtures.py --check
python Scripts/generate-statistics-ui-fixtures.py
deactivate
```

The first four commands compare independently generated references with the included fixtures. The statistics script prints reference values for review and has no `--check` mode. Its output therefore needs to be compared with the intended test cases separately.

The scientific generator uses direct MPFR calls; complex references use `mpmath`; financial, statistical, and conversion references use independent high-precision decimal or exact-factor calculations. Review the mathematical convention as well as the digits, especially for cash flow signs, angle units, branch choices, and rounding.

Take care with commands that omit `--check`: the scientific, complex, and conversion generators write fixture files by default. The financial generator prints by default and writes only with `--write`; the statistics generator prints its results. Review any generated diff as a source change. A generator mismatch calls for investigation before accepting new expected values.

D.13 Building this manual

Install a TEX distribution containing LuaLATEX, `latexmk`, and `makeindex`, together with the packages used by the manual. A full MacTEX installation is the straightforward macOS option: `https://tug.org/mactex/`. A suitable full TEX Live installation can also build the manual on Linux. The manual uses fonts provided by the TEX distribution and does not require the app's numeric libraries, Python environment, or an Xcode build.

Keep only one builder or watcher for the manual's output directory. Stop any existing watcher first, then run from the source root:

```
bash "SpicyRPN Manual/build.sh" --open
```

The resulting PDF is `SpicyRPN Manual/build/SpicyRPN-Manual.pdf`. Omit `--open` when no PDF viewer is wanted. On macOS, the Finder launcher `SpicyRPN Manual/Build Manual.command` provides the same build-and-open workflow. The build script finds its own source directory, so it can also be invoked from elsewhere using its full path. It adds the usual MacTEX binary directory to its search path when available and is compatible with the Bash supplied by macOS.

For continuous rebuilding while editing:

```
bash "SpicyRPN Manual/build.sh" --watch --open
```

Stop this process with Control-C. The script first requires a successful initial build before entering watch mode; errors during later edits are reported while the watcher waits for a correction.

The build uses `latexmk` with LuaLATEX, repeated passes as needed, and index-tool support. Shell escape is disabled, external `latexmk` configuration is ignored, and generated font caches are kept inside the manual's build directory. To remove generated manual files and rebuild, stop any watcher first. The `--clean` option

removes this manual's generated PDF, intermediate files, and font cache; keep source artwork outside that directory. It does not touch the app's root-level `build/` or math-library installations. Then run:

```
bash "SpicyRPN Manual/build.sh" --clean --open
```

The master source is `SpicyRPN Manual/SpicyRPN-Manual.tex`; chapter files, appendices, front matter, and back matter are included from their respective directories. Follow the existing style and metadata files when editing. After a successful build, inspect the affected PDF pages, page breaks, wide tables or command listings, links, and cross-references. A successful TEX exit alone does not prove that the page layout is readable. On failure, start with the first substantive error in `SpicyRPN Manual/build/SpicyRPN-Manual.log`.

D.14 Diagnosing a build problem

Work from the first meaningful error and the destination being built. Later linker or test errors may simply be consequences of an earlier missing dependency. The checks in table D.4 narrow the problem without changing calculator state.

Table D.4: Common build symptoms and the first useful checks.

Symptom	Check
Missing source archive	Restore the bundled GMP, MPFR, and MPC archives from the matching release ZIP. For a working copy without bundled MPC, check its download step or verified cache.
Archive checksum failure	Compare against the matching release's checksum file and restore the correct archive. Do not bypass the check to make the build continue.
SDK or developer-tool error	Inspect `xcode-select`, `DEVELOPER_DIR`, Xcode's installed platforms, and `xcodebuild -showsdks`. Use the intended full Xcode.
Missing GMP, MPFR, or MPC header or library	Build the libraries for the selected SDK and check its `build/install/` prefixes.
Wrong-platform linker error	Check macOS, simulator, and device separately. An arm64 archive from one platform cannot substitute for another.
Missing Mac architecture slice	Rebuild with the universal macOS script and check the `macos-universal` libraries. A Release archive must retain both arm64 and x86_64.

Symptom	Check
Library edit has no effect	Force the appropriate library rebuild, then clean and relink the app and launch the new executable.
App builds but iOS tests cannot run	Check the selected runtime or device OS against the common iOS 18.0 minimum, and check Xcode's installed platform and test support.
Signing or installation failure	Check the selected team, bundle identifiers, all relevant targets, device pairing, and Developer Mode using Xcode's diagnostic.
UI test launches but fails on input	Read the first failed activity; inspect the active sheet, keyboard focus, control accessibility, and exact destination before treating it as a numerical failure.
Manual command or package missing	Check for a complete, consistent TEX installation and the script's first diagnostic.
Manual cross-reference or layout problem	Let the build complete its passes, inspect the log and referenced label, then review the affected PDF pages.

When reporting a remaining problem, retain the source package version and local patch, tool versions, SDK and destination, exact command, and its first substantive error. Include any library-source or deployment-setting changes and, for tests, the result bundle and failing test identifier. Calculator operation and saved-state troubleshooting are covered in chapter 14.

Appendix E

MIT and GNU license texts

This appendix reproduces the complete MIT License for original SpicyRPN material, including this manual, and the GNU GPL version 3 and LGPL version 3 texts for the GNU libraries. The LGPL incorporates the GPL's terms with additional permissions for library use. The licensing and acknowledgments page identifies each component's license and version options.

The texts below retain their copyright and permission notices. Including them does not change any upstream component's licensing options.

E.1 MIT License

```
MIT License

Copyright (c) 2026 Bryan R. Davies

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

E.2 GNU General Public License, version 3

```
                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
```

use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

```
  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.

  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
```

```
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.

  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
```

```
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
```

```
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
```

source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law. If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it. (Additional permissions may be written to require their own
removal in certain cases when you modify the work.) You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or

    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10. If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term. If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

```
  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.

  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
```

organization, or substantially all assets of one, or subdividing an
organization, or merging organizations. If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License. For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based. The
work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version. For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement). To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients. "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that

country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to

address new problems or concerns.

  Each version is given a distinguishing version number. If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation. If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions. However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.

  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

```
  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <https://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<https://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<https://www.gnu.org/philosophy/why-not-lgpl.html>.
```

E.3 GNU Lesser General Public License, version 3

```
                   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.

  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:
```

```
   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.

  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.

   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
```

```
      system, and (b) will operate properly with a modified version
      of the Library that is interface-compatible with the Linked
      Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
   the Minimal Corresponding Source and Corresponding Application
   Code. If you use option 4d1, you must provide the Installation
   Information in the manner specified by section 6 of the GNU GPL
   for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based
   on the Library, uncombined with any other library facilities,
   conveyed under the terms of this License.

   b) Give prominent notice with the combined library that part of it
   is a work based on the Library, and explaining where to find the
   accompanying uncombined form of the same work.

  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```

Glossary and Mathematical Symbols

The terms below describe SpicyRPN as implemented. A key legend, a register name, and a mathematical variable can use the same letters with different meanings. The symbol tables following the alphabetical glossary distinguish these uses. For key sequences, see the command reference; for formulas and numerical limits, see the mathematical reference.

Alphabetical glossary

A

Absolute value. The nonnegative size of a real number. For a complex number, ABS returns its magnitude, $\sqrt{a^2 + b^2}$, rather than the absolute value of its real component. See complex functions (p. 90).

Accrued interest. The portion of a bond's next coupon attributable to the time from the previous coupon date to settlement. Clean price plus accrued interest is dirty price. See bond calculations (p. 160).

Accumulator. A running total summarizing statistical observations. The statistical bank holds a count and five sums, not the original observations; removing a point requires entering it again. See the six accumulators (p. 112).

Accuracy. Closeness to the intended mathematical or measured value. More working digits can reduce rounding error but cannot correct inaccurate inputs, an unsuitable model, or a coarser solver tolerance. See precision and accuracy (p. 46).

Actual day count. The difference between two Gregorian date ordinals. Consecutive dates differ by one day; the calculation does not add an extra day for the starting date. See date differences (p. 158).

Affine conversion. A conversion using both scale and offset, such as Celsius to Fahrenheit. Temperature conversions act on readings; an interval needs different handling because offsets do not apply to differences. See temperature conversions (p. 103).

ALG (algebraic entry). The mode in which operands and operators form an expression, with precedence and parentheses, evaluated by `=`. It retains its own unfinished expression and answer. Financial inputs use explicit storage and CPT; multi-input financial, statistical, and associative operations use labeled worksheets. See ALG mode (p. 31).

Amortization. Splitting loan payments into interest and principal while updating the balance and elapsed payment count. It differs from a TVM solve, which finds one unknown for the whole payment stream. See amortization (p. 134) and the ALG procedure (p. 137).

Angle unit. The shared DEG, RAD, or GRAD setting used for trigonometric arguments, inverse results, and polar angles. A full turn is 360°, 2π radians, or 400 gradians. Turn is also a conversion unit. See angle settings (p. 73).

Annuity. Equal payments separated by equal periods. END places each payment at its period's end; BEGIN places it at the beginning. The TVM register PMT is the signed payment amount. See payment timing (p. 126).

Annunciator. A short display indicator, called a flag in this manual, identifying a setting, pending command, active entry, or occupied bank. Reading it does not execute a command. See display flags (p. 215).

ANS. ALG's saved successful answer, retained when the expression is cleared. It is separate from the RPN stack and its Last X and Last Y operands. See answer updates and reuse (p. 39).

Arbitrary precision. Arithmetic with selectable working precision rather than a fixed hardware width. SpicyRPN accepts 16–10,000 decimal working digits and defaults to 600. Calculations still have finite precision and resource limits. See PREC (p. 47).

Arithmetic storage. Updating a memory value by adding, subtracting, multiplying, or dividing by the current operand through a STO modifier. It changes the target's stored value. See arithmetic storage (p. 60).

Associative memory. A bank addressed by numeric keys rather than register numbers. Numeric equality of the real components identifies keys, so equal real keys address one entry. See storage and recall (p. 63) and ALG worksheets (p. 64).

Associativity. The grouping rule for operators of equal precedence. ALG powers associate rightward: $2^{3^2} = 2^{(3^2)}$. Multiplication and division group leftward, including implicit multiplication. This parsing rule does not guarantee mathematical associativity of rounded arithmetic. See ALG precedence (p. 33).

Autosave. Saving state after actions without a separate Save command. The local database includes values, settings, unfinished entries, and pending prefixes. It is not cloud synchronization. See saved state (p. 170).

Avoirdupois. The mass convention used for ounces and pounds: 16 ounces make one pound, and one pound is exactly 0.45359237 kilograms. These are neither troy nor fluid-volume units. See mass conversions (p. 101).

B

Bank. A separately maintained collection of values: ordinary registers, associative memory, statistics, finance, or cash flows. Occupancy flags and clearing commands apply to these collections. Both entry modes share the banks. See memory banks (p. 57).

Basis, depreciable. Asset cost minus salvage value. The secondary depreciation result is remaining depreciable basis, excluding salvage: RPN Y or the ALG output Book value less salvage. It is not full book value. See depreciation inputs (p. 162).

BEGIN. Payment timing with the first regular TVM payment at time zero, identified by BGN. In amortization, only the first payment with elapsed count n=0 has no preceding interest period. See BEGIN and END (p. 126).

Binary operation. An operation requiring two operands. RPN uses Y op X; an ordinary binary result replaces the pair and drops the stack. Percentage functions instead retain Y. See stack effects (p. 16).

Bit. A binary digit. MPFR precision counts binary significand bits; a request for d decimal working digits becomes $\lceil d \log_2 10 \rceil$ bits. See numeric representation (p. 250).

Bond. A security represented here by regular semiannual coupons and redemption of 100 per 100 of face value. The app uses a maturity-anchored schedule and actual coupon-period day counts. See the supported bond model (p. 159).

Book value. Asset cost less accumulated depreciation, including retained salvage. Add salvage to the remaining-basis result to obtain book value. See depreciation results (p. 163).

Bracket, root. Lower and upper bounds whose function values have opposite signs, or whose endpoint is a root. An IRR bracket restricts the search and can select among multiple returns. See IRR bounds (p. 148).

Branch cut. A boundary across which a chosen single-valued complex function can be discontinuous. Logarithms, roots, and inverse functions require branch conventions; real and complex modes can return different valid roots. See principal branches (p. 94).

Build. A compiled application or the process producing it. A target defines what is built, a scheme selects actions and configuration, and a destination selects the Mac, device, or Simulator environment. See building the source (p. 272).

C

CANCEL. A control that dismisses a pending selection or an ALG worksheet's unaccepted edits. It does not undo accepted values or completed operations. See cancellation scope and controls (p. 194).

Cancellation, numerical. Loss of significant digits when subtracting nearly equal approximations. Increased precision can help, but stable formulas also matter. Composed calculations and custom special-function approximations can remain sensitive to cancellation even when each primitive is correctly rounded. See complex numerical limits (p. 96).

Cash flow. A signed payment or receipt at a specified time. In the cash-flow bank, CF_0 occurs at time zero and each expanded entry advances one equal period. Outflows are negative and receipts positive from the chosen viewpoint. See cash-flow timing (p. 140).

Cash-flow index. A zero-based position in the expanded cash-flow list. Register n selects the final included index for NPV and IRR, not the number of stored entries. SHOW has a separate inspection position. See calculation selection (p. 145).

Ceiling. The least integer greater than or equal to a real value. For example, $\lceil -1.2 \rceil = -1$. It differs from floor and nearest-integer rounding. See integer functions (p. 75).

Checksum. A value computed from file contents to check that a source or library archive matches an expected artifact. It identifies bytes rather than establishing program correctness. See source acquisition (p. 274).

Clean price. Bond price excluding accrued interest, per 100 of face value. PRC produces this amount and stores it in PV; YTM uses it as the price input. See price and yield (p. 160).

Clear. An operation whose scope depends on the command: X, an expression, the stack, or a selected bank. Stack clearing does not reset every bank or setting. A pending prefix can consume a clear key as cancellation. See clearing scope (p. 175).

Code signing. Attaching identity and integrity information to an application for its execution or installation environment. Device deployment can require a team and provisioning configuration. See signing and installation (p. 279).

Combination. A selection of r objects from n distinct objects without regard to order, counted by $\binom{n}{r} = n!/[r!(n-r)!]$. Inputs must be integers with $0 \le r \le n \le 10{,}000$. See counting functions (p. 79).

Commit, entry. Converting a numeric draft into a calculator value. Committing a number differs from evaluating an entire ALG expression or duplicating a stack value. See RPN ENTER behavior (p. 15) and ALG evaluation (p. 31).

Complex number. A single value $z = a + bi$ with real and imaginary components. Both components occupy one stack or memory position; rectangular and polar views represent the same value. See complex entry, representation, and mode behavior (p. 84).

Compound interest. Growth in which each period's interest contributes to the balance used in later periods. Fractional periodic rate r produces factor $(1 + r)^n$. TVM expects percentage $i = 100r$. See rates and periods (p. 125).

Constant recall. Repeatedly retrieving a value without changing its stored source. Ordinary-register RCL pushes the value in RPN and does not consume the register. See using a stored constant (p. 61).

Conversion family. Compatible units grouped by quantity, such as length, pressure, or energy. CONV selects source and destination within a family and returns a number; values acquire no persistent unit tags. See unit selection (p. 98).

Correlation coefficient. The dimensionless measure r of linear association returned with a regression estimate. It requires variation in both coordinates. A constant response can have a valid fitted line but undefined correlation. See estimation and correlation (p. 117).

Coupon. A bond's periodic interest payment. PMT stores the annual coupon percentage; each semiannual coupon is $PMT/2$ per 100 of face value. Coupon rate and yield are different inputs. See bond inputs (p. 159).

CPT. The ALG prefix for explicitly computing a financial result. It distinguishes solving from storing an input. See computation targets and procedure (p. 42).

Crossover depreciation. Declining balance that switches to straight line when distributing remaining basis over remaining life gives a larger deduction. The final period reaches salvage. See DB to SL (p. 164).

D

Database. The local SQLite store containing calculator state and value records. Saving is transactional; a failed save is handled by restoring the previous saved state where possible. See storage errors (p. 182).

Date order. MDY or DMY interpretation of numeric dates. In MDY, the month precedes `.DDYYYY`; in DMY, the day precedes `.MMYYYY`. Changing the setting changes interpretation, not stored digits. See date entry (p. 156).

Day-count convention. A rule translating dates or days into a financial interval. Actual days, the implemented 30/360 date rule, and simple interest divided by 360 or 365 are separate conventions. See day counts (p. 158).

Decimal place. A position after the decimal point. FIX, SCI, and ENG specify fractional places in the displayed mantissa; this differs from working digits. See display notation (p. 48).

Declining-balance depreciation. Depreciation based on current book value and rate $(F/100)/L$, where F is factor percentage and L useful life. Deductions respect salvage and remove remaining basis in the final year. See declining balance (p. 164).

Depreciation. Allocating an asset's depreciable cost over useful life. Each requested year is computed from cost, salvage, life, and method; the app does not maintain a cumulative depreciation ledger. See depreciation (p. 162).

Digit ruler. The CAL display aid marking positions in groups of five during long-number inspection. It counts digits, not their accuracy. See long-number inspection (p. 52).

Dirty price. Bond settlement price including accrued interest: clean price plus accrued interest. PRC returns clean price. See bond pricing (p. 160).

Discount rate. The rate translating future cash flows into present values. NPV uses percentage i per interval between cash-flow entries; the rate and interval must agree. See net present value (p. 146).

Display precision. The fractional-digit count requested for display notation. Formatting normally preserves the stored value; RND separately changes it. See displaying and rounding (p. 53).

Domain. Inputs on which a function is defined in the selected mode. Negative values are outside the real square-root domain but have principal complex square roots. See function domains (p. 82).

Drop. Removing X and shifting deeper values toward it. A fixed stack repeats its deepest value; an unlimited stack shrinks and retains zero X if otherwise emptied. See stack drop (p. 16).

E

Effective rate. Actual growth over an interval after compounding. With m periods per year and fractional periodic rate r, the effective annual rate is $(1+r)^m - 1$. See rate conventions (p. 125).

END. Timing with regular payments at times 1 through n. This is the default TVM timing; BGN is absent. See payment timing (p. 126).

Engineering notation. ENG display with an exponent divisible by three and one to three digits before the mantissa's decimal point. Fractional places do not specify a fixed significant-digit count. See SCI and ENG (p. 48).

Entry. The draft being edited. ENTRY means an active numeric draft in RPN and an entire unfinished expression in ALG. Drafts survive mode switches and saved-state restoration. See the ENTRY flag (p. 215).

Ephemeral test mode. A development configuration with nonpersistent state, protecting normal on-disk state from test changes. A uniquely named test database is another isolation mechanism. See test environments (p. 283).

Error function. The function $\operatorname{erf}(x) = 2\pi^{-1/2} \int_0^x e^{-t^2}\, dt$. Its complement is $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$; using erfc directly preserves small real-tail values better than subtraction. See erf and erfc (p. 81).

Exponent. The power in a^b, or the power of ten in a literal such as 1.2×10^5. EEX edits the latter; it is not the general power operator. See scientific-number entry (p. 22).

Expression. Operands, operators, groups, and function calls representing a calculation. ALG retains this structure until evaluation; RPN executes operations on available operands. See ALG expressions (p. 31).

F

Factorial. For a nonnegative integer n, $n! = 1 \cdot 2 \cdots n$, with $0! = 1$. Real-mode factorial accepts integers through 10,000. Complex mode extends it through $\Gamma(z+1)$, including noninteger real inputs, subject to poles and numerical limits. See factorials (p. 79).

Financial register. One of n, i, PV, PMT, or FV. Financial families reuse them with different meanings; check retained values before a calculation. See financial registers (p. 124) and the context table below.

Fixed-point display. FIX formatting with a selected number of fractional places. Scientific fallback can occur when fixed formatting cannot usefully show the value. It is a display choice, not fixed-point arithmetic. See FIX (p. 50).

Fixed stack. An RPN stack with a configured position count. Default depth is five; the UI accepts 2–1,000. Lifting can discard the deepest value; dropping repeats it. See stack depth (p. 20).

Fixture, test. Known inputs, expected results, and setup for a repeatable test. Oracle fixtures can supply expectations calculated outside the application. See oracle data (p. 287).

Flag. An annunciator reporting state. MEM, ASSOC, STAT, FIN, and CF identify occupied banks, including stored zero; ENTRY and prefix flags identify unfinished work. Occupancy does not certify a complete or valid problem. See all flags (p. 215).

Floating-point number. A finite-precision significand with an exponent, permitting a wide magnitude range. MPFR uses binary significands; many finite decimal fractions therefore require rounding. See number representation (p. 250).

Floor. The greatest integer no larger than a real value. For example, $\lfloor -1.2 \rfloor = -2$, whereas truncation gives -1. See integer functions (p. 75).

Fractional part. The signed remainder after truncation: $\mathrm{FRAC}(x) = x - \mathrm{INT}(x)$. Thus FRAC of -1.2 is -0.2. See INT and FRAC (p. 75).

FUNC. The ALG prefix making a selected function start an explicit call, with groups or comma-separated arguments where applicable. See function-call entry (p. 36).

Future value. The signed terminal TVM amount at time n. The FV register instead holds salvage value for depreciation. See TVM registers (p. 124).

G

Gamma function. The factorial extension satisfying $\Gamma(n+1) = n!$ for nonnegative integers, with poles at zero and negative integers. Real log-gamma is not a general $\ln|\Gamma(x)|$ command. See gamma and log-gamma (p. 80).

GMP. The GNU Multiple Precision Arithmetic Library, used beneath MPFR. Building the app requires appropriate library artifacts for each target platform. See numeric libraries (p. 276).

Gradian. One hundredth of a right angle; a full turn contains 400 gradians. GRAD selects this interpretation for trigonometric and polar angles. See angle units (p. 73).

Gregorian calendar. The calendar used for dates, extended backward with the same leap-year rules throughout years 1–9999. Numeric dates have no time zone or time of day. See calendar dates (p. 156).

Guard digits. Extra intermediate precision reducing rounding loss. They improve numerical calculations without making approximations exact or removing domain limits. See complex arithmetic (p. 254).

H

Hyperbolic functions. The exponential-based sinh, cosh, and tanh functions and their inverses. HYP and INVHYP select them through trigonometric keys. Their arguments are not interpreted as DEG, RAD, or GRAD angles. See hyperbolic functions (p. 75).

I

Imaginary unit. The number i satisfying $i^2 = -1$, used in $a + bi$. It is unrelated to financial i, which holds a real percentage rate or depreciation factor. See complex components (p. 85).

Immutable value. A value whose stored components and creation precision do not change in place. Raising precision affects subsequent calculations and cannot reconstruct digits lost earlier. See existing values (p. 54).

Implicit multiplication. ALG multiplication inserted between adjacent compatible operands or groups. It has the same precedence as explicit multiplication and division. Use parentheses for another intended grouping. See implicit multiplication (p. 34).

Infix. Notation placing operators between operands, as in $2 + 3$. ALG uses infix expressions. Separately, INFIX in precision help means working decimal digits; it does not select ALG. See precision terminology (p. 47).

Integer part (INT). Truncation toward zero. Scientific INT, on orange-shift ENTER in RPN or orange-shift $1/x$ in ALG, differs from blue-shift financial i, whose INT legend means simple interest. See integer and fractional parts (p. 202).

Interest rate. A proportion charged, earned, or used for discounting over an interval. Financial i uses percentages: enter 5 for 5%. Its period is context dependent: periodic for TVM and NPV, annual for simple interest and bond yield. See rate units (p. 125).

Intermediate result. A value calculated for later use in a larger calculation. The RPN stack retains several such values; ordinary memory can preserve them beyond subsequent stack operations. See intermediate results (p. 25).

Internal rate of return (IRR). A periodic percentage at which selected cash flows have zero NPV. There may be no root, one, or several; a guess or bracket influences the search. The result does not automatically replace i. See IRR (p. 147).

Inverse function. A function reversing another on a chosen domain or branch. Inverse trigonometry returns an angle, not the reciprocal of the corresponding trigonometric value. See inverse trigonometry (p. 73).

Iteration. One repeated step of an algorithm. Financial solvers have iteration, evaluation, and tolerance limits independent of working-digit count. See solver policy (p. 263).

K

Key. Either a keyboard control or, in associative memory, the numeric value identifying a stored item. A shifted legend selects an operation; an associative key is the real component of a numeric value, in either complex setting. See shifted keys (p. 8) and numeric keys (p. 63).

L

Last X and Last Y. Separate RPN registers preserving earlier operands for recovery and reuse. They are not the current X and Y positions, shown as S0 and S1, and do not form an undo history. See which operations save and recall operands (p. 19).

LaTeX. The source-document system for this manual. LuaLaTeX typesets it, and latexmk coordinates passes and index processing. The calculator is built separately. See manual builds (p. 288).

Least squares. Fitting a model by minimizing squared residuals. SpicyRPN's ordinary unweighted fit is $y = a + bx$. See linear regression (p. 116) and ALG statistical outputs (p. 120).

Lift. Moving stack values deeper to make room at X. Starting an RPN number lifts once; committing it does not lift again. A fixed stack can lose its deepest value. See stack lift (p. 16).

Linear conversion. Conversion by scale without offset, such as meters to feet. Exact decimal or rational conversion factors are still evaluated at finite working precision. See conversion arithmetic (p. 107).

Logarithm. The exponent needed to obtain a value from a base. LN uses base e; LOG uses base 10. RPN's arbitrary-base logarithm takes base Y and argument X. See logarithms (p. 72).

M

Magnitude. The nonnegative radial component $|z|$ of a complex value, paired with phase in polar display and returned by ABS. It generally differs from the real component. See polar representation (p. 88).

Mantissa, display. The coefficient before the power of ten in SCI or ENG, such as 1.234 in 1.234×10^6. The stored binary significand is a different representation. See number notation (p. 48).

Maturity. A bond's final payment date. Preceding semiannual coupon dates are derived backward from maturity, preserving its month-end convention. See bond dates (p. 159).

MCLR. A prefix that selects a group to clear, rather than an individual memory address. See selectors, clearing scope, and cancellation (p. 204).

Mean, arithmetic. Observation sum divided by count. Paired data has a separate mean for each coordinate; this differs from the weighted mean. See means and their outputs (p. 114).

Memory occupancy. The presence of stored data, including explicit zero. Recalling an unset register returns zero without occupying it. A zero display therefore does not establish whether a bank is empty. See empty and zero (p. 62).

Mode. A setting selecting behavior, such as RPN/ALG, real/complex, or BEGIN/END. Their scopes differ; switching entry mode preserves each workspace. See shared and independent state (p. 65).

Modulo and remainder. Operations subtracting an integer multiple of a divisor. SpicyRPN distinguishes truncating remainder from IEEE-style remainder using a nearest-even quotient. Neither implies an always-nonnegative mathematical modulo. See remainder conventions (p. 77).

MPC. The GNU Multiple Precision Complex library, built on GMP and MPFR, used for SpicyRPN's complex arithmetic and elementary functions. Each primitive's real and imaginary results are individually rounded; composed calculations and custom special functions retain their own numerical limits. See complex arithmetic (p. 254) and MPC build setup (p. 278).

MPFR. The multiple-precision floating-point library used for real arithmetic and many real functions. Precision is specified in bits; ordinary arithmetic rounds to nearest with ties to even. See MPFR representation (p. 250).

N

NaN. "Not a Number," representing an undefined numerical result. It is not a valid ordinary financial input or real component of an associative key. Invalid operation results commonly become displayed errors. See calculation errors (p. 180).

Net present value (NPV). Selected cash flows discounted to time zero at percentage i, including undiscounted CF_0. Register n selects the final included index. See NPV (p. 146).

Nominal annual rate. An annual percentage quoted as a multiple of a periodic rate. For monthly compounding, divide the nominal annual percentage by 12; this does not convert an effective annual rate. See rate conversion (p. 125).

Numeric literal. An entered number's characters, including decimal and optional exponent. Drafts can preserve trailing zeros and unfinished exponents before conversion to values. See numeric entry (p. 22).

O

Operand. A value an operation acts on. RPN unary functions use X; binary functions use Y followed by X. ALG precedence and grouping determine operands. See operand order (p. 16).

Oracle, test. An independent source of expected results, such as a separate high-precision implementation and its fixtures. Agreement is evidence subject to matching conventions and inputs. See financial oracles (p. 287).

Ordinary register. One of ten general-purpose memory locations R0–R9. They are separate from the current stack positions X (visible S0) and Y (visible S1), and from Last X and Last Y. See ordinary storage and recall (p. 58).

P

Packed date. A number whose integer and six fractional decimal positions encode calendar fields under MDY or DMY. It has no attached date type; ordinary numeric arithmetic is not calendar arithmetic. See date encoding (p. 156).

Payment. The signed equal periodic cash flow in PMT for TVM and amortization. For bonds, PMT instead contains an annual coupon percentage. See payment registers (p. 124).

Percent. A proportion per hundred. RPN `%` computes $YX/100$, retaining Y; percentage change and percentage of total use different formulas. In worksheet field text, infix `%` instead denotes truncating remainder. See percentage functions (p. 78).

Period. The equal interval in a financial calculation. Rate, payment spacing, and period count must agree. No calendar unit is attached to a stored count. See period consistency (p. 125).

Permutation. An ordered selection of r objects from n, counted by $n!/(n-r)!$. Inputs must be integers with $0 \le r \le n \le 10{,}000$. See permutations and combinations (p. 79).

Persistence. Retaining state across launches, including both entry workspaces and shared banks and settings rather than only the visible answer. See resuming work (p. 173).

Phase. A complex value's polar angle, also called its argument. Display follows the angle unit; zero has phase zero by convention. See polar angles (p. 88).

Polar representation. Magnitude and phase, $r\angle\theta$, equivalent to $r(\cos\theta + i\sin\theta)$. Stored components remain rectangular. See polar entry and display (p. 88).

Pole. A singularity where a function becomes unbounded, such as gamma at a nonpositive integer. Complex mode does not make a pole a finite valid result. See gamma's domain (p. 80).

Postfix. Placing an operation after its operands, as in RPN arithmetic. Many ALG scientific keys instead act postfix on the trailing operand or closed group; FUNC starts an explicit call. See ALG functions (p. 35).

Precedence. Operator priority in ALG grouping. Parentheses override it; multiplication and division precede addition and subtraction. See precedence rules (p. 33).

Precision. Digits carried in a representation or calculation. Working precision, display places, and algorithmic stopping accuracy differ. More displayed digits are not necessarily more meaningful digits. See three precision concepts (p. 46).

Prefix, command. A pending selection modifying subsequent keys, such as STO, RCL, HYP, FUNC, CPT, or MCLR. It is distinct from prefix mathematical notation and unit prefixes such as kilo. See canceling prefixes (p. 174).

Present value. The signed TVM amount at time zero, in PV. Other families reuse PV for balance, negative simple-interest principal, bond clean price, or asset cost. See financial registers (p. 124).

Primary result. The designated ALG output that becomes ANS after a successful computation. Selecting a companion output does not redefine the primary result or change ANS. See worksheet outputs and transfers (p. 42).

Principal branch. The chosen standard value of a multivalued complex function. On negative real input, SpicyRPN's logarithm uses imaginary part $+\pi$, and square root uses the positive imaginary root. See branch conventions (p. 94).

Principal, financial. Capital excluding interest. Simple interest uses $P = -PV$; amortization separates principal from interest. This meaning is unrelated to a principal complex branch. See simple interest (p. 155).

Provenance. Internal records linking a value to its operation and sources. They support storage and development inspection, rather than providing user-facing undo or calculation history. See stored state (p. 170).

R

Radian. The angle subtending an arc equal in length to its radius. A full turn is 2π radians. Complex exponential and logarithmic formulas use radian-valued imaginary arguments independently of display angle units. See complex conventions (p. 90).

Real-only operation. An operation defined on real components, including finance, statistics, conversions, and several integer functions. A nonzero imaginary input

in complex mode can produce “Used real portion only.” See real-only functions (p. 93).

Recall. Retrieving a stored value without consuming its source. Insertion, stack movement, and Last-register effects depend on the memory bank and entry mode. See ordinary recall (p. 58) and specialized banks (p. 66).

Reciprocal. The inverse $1/x$, defined for nonzero x. It is unary, unlike exchanging a division’s two operands. See powers and reciprocal (p. 71).

Rectangular representation. Real and imaginary components a, b of $a + bi$. Component Swap exchanges these components even in polar display. See component swapping (p. 90).

Redemption value. The amount repaid at bond maturity, fixed here at 100 per 100 of face value. FV does not alter this amount. See bond assumptions (p. 159).

Register. A named value location. Current RPN X and Y appear as S0 and S1; ordinary R0–R9, saved Last X and Last Y, and financial registers are separate locations. See register names (p. 13).

Regression. Estimating a relationship from observations. SpicyRPN supports the ordinary linear fit $y = a + bx$ from paired accumulators and can return fitted $\hat{y}$ and correlation. See linear regression (p. 116).

Repetition count, cash flow. The total equal consecutive occurrences specified by Nj, including the one already entered by CFj. Storage expands repetitions into period entries. See repeated flows (p. 142).

Residual. The difference between a model result and its target. In regression, observed Y minus fitted Y; in a financial solve, the remaining equation value that should be zero. See financial solver policy (p. 263).

Roll. A cyclic rearrangement of a fixed stack, up or down, without arithmetic. Unlimited stacks do not support roll. See stack rearrangement (p. 18).

Root. Either an indexed root, such as a cube root, or a solution of $f(x) = 0$. Complex indexed roots use principal values; financial solvers seek real roots in allowed domains. See indexed roots (p. 71) and equation solving (p. 263).

Rounding. Replacing a value with one representable at a selected precision or increment. Ordinary MPFR arithmetic, display, and NINT use nearest with ties to even. Financial RND rounds to decimal places with ties away from zero. See rounding rules (p. 53).

RPN (Reverse Polish Notation). Entry in which operands precede the operation using them. The stack retains values and intermediate results, letting the user choose evaluation order without expression parentheses. See learning RPN (p. 226).

S

Salvage value. The residual asset value retained after depreciation, in FV. Depreciable basis excludes it, and methods limit depreciation so salvage is not consumed. See asset inputs (p. 162).

Sample, statistics. An entered pair of X and Y values; use Y equal to zero for single-variable work. Count records observation updates, not total Y weights. Original observations are not retained as a list. See statistical entry (p. 111).

Scientific notation. SCI display with one digit before the decimal point and an explicit power of ten. For nonzero values, d fractional places normally show $d + 1$ significant digits. See display formats (p. 48).

SDK. A Software Development Kit providing platform headers, libraries, and tools. Mac, iOS device, and Simulator remain different destinations even when CPU architectures coincide. See build prerequisites (p. 272).

Settlement. The bond valuation date, which must precede maturity. A coupon due on settlement is excluded from the future-payment schedule. See settlement conventions (p. 160).

Shift. A one-operation keyboard layer selected by orange f or blue g, identified by its flag. It selects a legend rather than multiplying by f or g. See shift controls (p. 8).

Sign convention. A consistent financial viewpoint: outflows are negative and receipts positive. Changing viewpoint reverses all relevant signs, not just one amount. See cash-flow signs (p. 125).

Significand. Precision-bearing floating-point digits, separate from the exponent. Stored significands are binary although entry and display are decimal. See number representation (p. 250).

Significant digits. Digits expressing numerical detail, counted from the first nonzero digit. They differ from places: 123.45 has five significant digits and two fractional places. See digit counts (p. 48).

Simple interest (INT). Interest proportional to principal, annual rate, and elapsed days without compounding. Blue-shift financial i returns 360-day and 365-day basis amounts. This INT differs from integer part on orange-shift ENTER in RPN or orange-shift $1/x$ in ALG. See simple interest (p. 155).

Simulator. Apple's environment for running iOS applications on a development Mac. A Simulator binary is not interchangeable with a device binary merely because both are arm64. See iOS destinations (p. 281).

Solver. A procedure calculating an unknown from inputs. TVM solves one register using all four others, including unwritten zeros; rates can require iterative root finding. See store and solve (p. 127).

SQLite. The embedded database engine for local persistence. The app uses transactions and write-ahead logging to maintain saved state. See source organization (p. 275).

Stack. The ordered RPN collection with current X at the working end, displayed as S0, and Y next, displayed as S1. Deeper positions follow. The stack is separate from memory banks and from the saved Last X and Last Y operands. See the RPN stack (p. 13).

Stack inspector. The read-only full-stack view opened from the display triangle, preserving entries and pending prefixes during inspection. See inspecting the stack (p. 17).

Standard deviation. Dispersion measured in the original data units. SpicyRPN returns sample deviations using denominator $n-1$, requiring at least two observations. See sample deviation (p. 116).

Standard gravity. The defined acceleration $g_0 = 9.80665\,\mathrm{m/s^2}$, a fixed conversion reference rather than measured local gravity. See mechanical units (p. 104).

Static library. Compiled code linked into an application during build. It must match both platform and architecture; arm64 device and Simulator libraries differ. See library deployment (p. 276).

Straight-line depreciation. Allocating basis evenly over useful life, with a fractional final year receiving a fractional annual deduction. See SL and SYD (p. 163).

Sum-of-years'-digits depreciation. Allocating basis with descending life weights. Fractional life retains fractional weights, such as 2.5, 1.5, and 0.5, instead of first rounding life to three. See fractional life (p. 166).

SwiftUI. Apple's interface framework used by SpicyRPN. The source directory named Preview contains the live keyboard and display implementation. See project layout (p. 275).

T

Ties to even. Nearest rounding that selects an even final retained digit at an exact halfway case. NINT maps both 1.5 and 2.5 to 2. See rounding conventions (p. 251).

Time value of money (TVM). The relationship among periods, periodic interest, present value, equal payments, and future value. The five registers describe regular payments rather than arbitrary cash-flow sequences. See TVM calculations (p. 124).

Tolerance. A threshold stopping an iterative calculation, such as interval width or residual size. It is independent of display places; 600 working digits do not imply a 600-digit solver stopping threshold. See solver tolerances (p. 263).

Transaction, database. Storage changes saved together or rolled back together, helping prevent partially updated calculator state from becoming the saved state. See save failures (p. 182).

Truncation. Removing the fractional part toward zero. Unlike floor, it moves negative nonintegers toward zero. Scientific INT and the quotient in truncating remainder use this rule. See integer operations (p. 75).

U

Unary operation. An operation using one operand, such as reciprocal. In RPN it normally replaces X without consuming Y; in ALG it normally acts on the trailing operand or group. See unary functions (p. 16).

Unit alias. A short conversion label such as `lb`, `US gal`, or `Imp gal`. Qualifiers matter: `oz` means mass, while `US fl oz` means volume. See the unit-symbol table.

Unlimited stack. The depth-zero RPN setting, growing and shrinking as needed within available resources. It retains a zero X when otherwise empty and does not support roll. See stack settings (p. 20).

Useful life. The positive depreciation duration stored in `n`. It may be fractional; requested years must still be integers beginning at one. See fractional life (p. 166).

V

Variance. The square of standard deviation. Sample variance divides the sum of squared deviations from the mean by $n - 1$. Positive X variation is required for a nondegenerate fitted line. See dispersion (p. 116).

W

Weighted mean. The value $\sum xy / \sum y$, treating each Y as its paired X's weight. Weights do not alter count or make other statistical functions frequency-weighted. See weighted mean (p. 114).

Working precision. Precision for new values and calculations, selected through PREC and shared across entry modes. Existing values retain their creation precision. See working digits (p. 47).

Worksheet. An ALG panel with named inputs and outputs, separate from the main expression. Accepted inputs and unaccepted field drafts are distinct. See evaluation, acceptance, computation, and result transfer (p. 42).

Workspace, entry. One mode's independent state: RPN stack and draft, or ALG expression and answer. Memory banks and many settings are shared. See switching workspaces (p. 171).

X–Z

Xcode. Apple's environment for building and running the app and tests. Scheme, configuration, destination, and installed SDKs affect the build. See development tools (p. 272).

Yield to maturity. The annual nominal percentage implied by clean bond price, coupons, redemption, and dates under the semiannual model. It differs from the bond's coupon percentage. See bond price and yield (p. 160).

Zero, stored and signed. A number that can occupy memory. Floating-point values can also distinguish positive and negative zero, affecting some branch-boundary angles. Neither distinction means an empty bank contains stored data. See stored zero (p. 62) and signed zero (p. 94).

Mathematical symbols by context

Symbols are local to their formulas. Italic mathematical letters are not automatically register names or commands. In particular, i, n, r, P, and N have several meanings.

Entry, stack, and overloaded notation

Symbol or notation	Meaning in this context
X, Y	Current RPN values, displayed as S0 and S1 respectively. Y is normally the left binary operand, X the right. Statistical variables x, y and ALG fields labeled X and Y identify inputs, not stack positions.
S0, S1, S_k	Visible RPN stack positions: S0 holds current X, S1 holds current Y; indices increase toward deeper values.
R0–R9	Ordinary memories. R4–R9 can receive a statistical snapshot through RCLΣ.
LAST X, LAST Y	Recall earlier operands from separate saved registers, not the current S0/S1 positions or the complete prior state.
ANS, =	ALG's saved successful answer and expression evaluation. Repeated equals does not repeat the last binary operation.

Continued on the next page

Symbol or notation	Meaning in this context
i, i	Imaginary unit in complex mathematics; a real percentage rate or factor in financial registers.
n, n	Generic integer or statistical count in mathematics; a financial register reused for periods, elapsed payments, cash-flow index, days, or useful life.
INT	Integer part at orange-shift ENTER in RPN or orange-shift $1/x$ in ALG; simple interest at blue-shift financial i.
%	Percentage $YX/100$, retaining Y in RPN. In worksheet field text, infix % instead denotes truncating remainder.
$\Delta\%$, T%	Percentage change $100(X-Y)/Y$ and percentage of total $100X/Y$. Both retain Y in RPN and require nonzero Y.
Δ	A mathematical difference. In Δ**FIX**, it identifies ALG's decimal-place prompt.
f, g	Mathematical function names; orange and blue shifts when written f and g. Unit g is gram.
k	An index or count in formulas; as a unit prefix, k means 10^3.

Arithmetic, functions, and precision

Symbol or notation	Meaning in this context
$=, \approx, \neq$	Exact equality, approximation, inequality. A printed approximation does not assert every shown digit is exact.
$<, \leq, >, \geq$	Order comparisons. An interval $[a, b]$ includes endpoints; (a, b) excludes them.
∞, NaN	Infinity and Not a Number; neither is an ordinary finite financial input.
$\lvert x\rvert$, $1/x$	Absolute value and reciprocal; $\lvert z\rvert$ is complex magnitude.
$x^2, \sqrt{x}, y^x$	Square, principal square root, and power. RPN y^x uses base Y and exponent X.
$\sqrt[x]{y}$	Indexed root of Y with index X. Real negative radicands need an allowed odd integer index; complex mode selects a principal root.
e, π	Natural-logarithm base and circle constant, computed at working precision.
$\exp x, \ln x, \log_{10} x$	Exponential, natural logarithm, common logarithm. Generally $\log_b a = \ln a / \ln b$ on the appropriate domain or branch.
$\sin, \cos, \tan$	Circular trigonometric functions, using angle settings at the points described in the entry-mode chapters.

Continued on the next page

Symbol or notation	Meaning in this context
$\sin^{-1}, \cos^{-1}, \tan^{-1}$	Inverse functions, also arcsin, arccos, arctan; the superscript does not mean reciprocal here.
$\operatorname{atan2}(y, x)$	Quadrant-aware angle of real pair (x, y). Operand order matters.
$\sinh, \cosh, \tanh$	Hyperbolic functions; asinh, acosh, atanh are inverses. Arguments are not angle-unit quantities.
$\lfloor x \rfloor, \lceil x \rceil$	Floor and ceiling. $\operatorname{trunc}(x)$ instead removes the fraction toward zero.
$n!, \binom{n}{r}$	Factorial and combinations. Permutations are $n!/(n-r)!$; RPN supplies n in Y and r in X.
$\Gamma(z), \ln\Gamma(z)$	Gamma and log-gamma. For nonreal inputs, complex log-gamma takes the principal logarithm of the computed gamma.
$\operatorname{erf}, \operatorname{erfc}$	Error function and complement, related by $\operatorname{erfc}(x) = 1 - \operatorname{erf}(x)$.
d, p	Requested decimal working digits and binary precision $p = \lceil d \log_2 10 \rceil$. Elsewhere d can mean display places or days.
ϵ, tolerance	A locally defined error bound or stopping threshold, not automatically 10^{-d} for working precision.
$\sum, \prod$	Sum and product over the specified terms or indices.

Complex numbers

Symbol or notation	Meaning in this context
$z = a + bi$	Real component a, imaginary component b, together one value.
$\Re z, \Im z$	Real and imaginary components. The interface's Re/Im selects a component, not a second stack position.
$\bar{z}$	Conjugate $a - bi$, a mathematical notation that does not imply a dedicated keyboard command.
$r = \lvert z \rvert$	Magnitude $\sqrt{a^2 + b^2}$, distinct from financial fractional rate or statistical correlation.
$\theta, \arg z, \operatorname{Arg} z$	Phase or argument. Capitalized Arg denotes a chosen principal value, with the complex chapter's branch conventions.
$r\angle\theta$	Polar representation; entry becomes rectangular using the angle unit at commit.
$\operatorname{Log} z$	Principal logarithm $\ln \lvert z \rvert + i \operatorname{Arg} z$; its imaginary angle is in radians regardless of display angle setting.

Continued on the next page

Symbol or notation	Meaning in this context
z^w	Generally $\exp(w \operatorname{Log} z)$, with an integer-power path where applicable.

Statistics and regression

Symbol or notation	Meaning in this context
x, y	Observation coordinates, or value and weight for weighted mean. RPN puts x in X and y in Y; ALG uses fields labeled X and Y.
n	Observation count, separate from financial n.
$\Sigma x, \Sigma y$	Coordinate sums. Keys $\Sigma+$ and $\Sigma-$ add or subtract an observation.
$\Sigma x^2, \Sigma y^2, \Sigma xy$	Sums of squares and paired products, not squares or products of sums.
$\bar{x}, \bar{y}, \bar{x}_w$	Coordinate means and weighted mean $\bar{x}_w = \Sigma xy / \Sigma y$.
s_x, s_y	Sample standard deviations; squared, they are sample variances. Their denominator uses count minus one.
a, b	Intercept and slope of $\hat{y} = a + bx$. RPN L.R. returns b in X and a in Y; ALG labels them Slope and Intercept.
$\hat{y}, r$	Predicted Y and correlation, returned in RPN X and Y or under separate ALG output labels.
M, N, P	Centered, count-scaled sums: $M = n\Sigma x^2 - (\Sigma x)^2$, $N = n\Sigma y^2 - (\Sigma y)^2$, $P = n\Sigma xy - \Sigma x \Sigma y$. Capital N is the Y moment, not observation count.

Financial registers, TVM, and simple interest

These are context-dependent meanings, not permanent types attached to the registers. Check the selected problem's inputs before calculation.

Symbol or notation	Meaning in this context
n	TVM: payment periods. AMORT: payments already processed. NPV/IRR: last included cash-flow index. Simple INT: days. Depreciation: useful life.
i	TVM, AMORT, NPV: percentage per period. IRR: percentage guess when supplied. Simple INT: annual percentage. Bonds: annual nominal yield percentage. DB methods: factor percentage.

Continued on the next page

Symbol or notation	Meaning in this context
PV	TVM: signed present value. AMORT: signed balance. Simple INT: negative principal. Bonds: clean price per 100. Depreciation: cost.
PMT	TVM and AMORT: signed regular payment. Bonds: annual coupon percentage. Not a cash-flow-bank entry.
FV	TVM: signed terminal amount. Depreciation: salvage. It does not alter bond redemption of 100.
$r = i/100$	Fractional periodic rate for TVM, amortization, and NPV. A 5% rate is stored as 5, not 0.05.
$q = 1 + r$, q^n	One-period and full-term TVM growth factors. The ordinary annuity factor is $(q^n - 1)/r$, with limit n at zero rate.
b, A_f, A_p	Timing indicator $b = 0$ for END, $b = 1$ for BEGIN; future and present annuity factors $A_f = q^b(q^n - 1)/r$, $A_p = A_f q^{-n}$, with zero-rate limits n.
k	Requested payment count for an AMORT block, entered in RPN X or the ALG Payments to apply field, distinct from elapsed payments already in n.
I_j, D_j, B_j	Amortization interest portion, principal portion, and balance after payment j. Initial balance is $B_0 = PV$; elapsed count starts at n_0.
P	In amortization, signed payment $P = PMT$. In simple interest, initial principal $P = -PV$.
d, j	In simple-interest formulas, days $d = n$ and annual fractional rate $j = i/100$. Here j is not a payment index.
I_{360}, I_{365}	Simple interest $P(i/100)n/360$ and $P(i/100)n/365$. These fixed denominators are not an actual/actual leap-year convention.

Cash flows and numerical rate solving

Symbol or notation	Meaning in this context
CF_0, CF_j	Cash flows at time zero and later integer period j. CFj appends the next entry in RPN; in ALG it opens the corresponding worksheet for confirmation with APPEND.
N_j, Nj	Total repetitions of the last CFj amount, including the entered occurrence; storage expands them.
N	Final included index in $\sum_{j=0}^{N} CF_j(1 + r)^{-j}$, selected by financial n. The sequence has $N + 1$ included entries.

Continued on the next page

Symbol or notation	Meaning in this context
NPV(r)	Discounted sum as a function of fractional rate; keyboard input uses percentage i.
IRR	A percentage $100r$ satisfying NPV(r) $= 0$, not an amount of money.
L, U, bounds	Lower and upper search endpoints as locally defined. The IRR dialog accepts percentage endpoints requiring an appropriate bracket.
$t = \ln(1 + r)$	Transformed rate used by the solver, enforcing $r > -1$, or percentage above -100.
$f(t)$	Transformed, possibly scaled residual. A small residual and a small bracket width are different measures.

Dates and bonds

Symbol or notation	Meaning in this context
MDY, DMY	Month/day/year and day/month/year interpretations of packed dates, not conversion commands.
Y, M, D	Calendar year, month, day in date formulas. Calendar Y differs from stack Y and a statistical Y observation.
ΔD, $D_2 - D_1$	A day difference under the specified actual or 30/360 convention; do not subtract packed numeric dates directly.
$C = PMT/2$	Semiannual coupon per 100 face value. PMT holds annual coupon percentage.
$r = i/200$	Fractional semiannual yield. Financial i holds annual nominal percentage, unlike TVM's periodic percentage.
N	Remaining coupon count, excluding a coupon on settlement. It is not read from financial n.
E, d, w	Actual days in the surrounding coupon period, days from settlement to the next coupon, and remaining fraction $w = d/E$.
$A = C(1 - w)$	Accrued interest; zero when settlement is a coupon date under the implemented schedule.
$P_{\text{clean}}, P_{\text{dirty}}$	Prices per 100 excluding and including accrual: $P_{\text{dirty}} = P_{\text{clean}} + A$.
100	Fixed redemption per 100 face value, not a value read from FV.

Depreciation

Symbol or notation	Meaning in this context
K, S, L	Cost, salvage, useful life in PV, FV, and `n`. Unit tokens `K` and `L` have different meanings.
$B = K - S$	Initial depreciable basis, excluding salvage.
j, m	Requested integer year j, starting at one, and total year slots $m = \lceil L \rceil$ for fractional life.
u_j	Fraction of a year in slot j, limited by remaining life. SL applies it to the final partial year.
D_j, R_j	Year's depreciation and remaining basis, returned in RPN X and Y or under separate ALG output labels. Book value is $S + R_j$.
V_h, B_h, L_h	Book value after year h, basis before that year, and remaining useful life. Unlike amortization balance B_j, depreciation B_h excludes salvage.
W, U_j	Sum of all SYD weights and cumulative weights through year j: $W = mL - m(m-1)/2$, $U_j = jL - j(j-1)/2$.
F, q	Factor percentage F, stored in `i`, and annual fraction $q = (F/100)/L$. Double declining balance uses $F = 200$.
SL, SYD	Straight-line and sum-of-years'-digits methods.
DB, DB→SL	Declining balance and crossover to straight line. The final active year removes remaining basis.

Conversion symbols and aliases

These labels identify conversion choices; typing letters does not attach units to values. Prefix case matters: kilo `k` is 10^3, mega `M` is 10^6, giga `G` is 10^9, centi `c` is 10^{-2}, milli `m` is 10^{-3}, micro μ is 10^{-6}, and nano `n` is 10^{-9}. The interface's micro sign `µ` has the same unit meaning as Greek μ. Bare `m` means meter; `mm` means millimeter. The table below distinguishes easily confused symbols. Every selectable unit appears in the complete unit inventory (p. 218); factors appear in the mathematical reference (p. 268).

Symbol or notation	Meaning in this context
`g`, g_0	Gram is a mass unit; standard gravity is an acceleration. Neither is the blue shift `g`.
`oz`, `fl oz`	Ounce without "fl" is avoirdupois mass; fluid ounce is volume and needs its US or Imperial qualifier.

Continued on the next page

Symbol or notation	Meaning in this context
`US`, `Imp`	Distinct liquid-volume conventions, not interchangeable prefixes. US volume choices are liquid measures, not dry measures.
`lb`, `lbf`	Pound mass and pound-force. Converting between them requires a physical acceleration assumption, not a change of unit within one family.
`t`, short ton, long ton	Metric tonne, US short ton, and Imperial long ton are three different mass units.
`K`, `L`, `N`	Kelvin, liter, and newton as unit symbols. The same letters can denote cost, life, or a count or moment in mathematical formulas. Kelvin has no degree sign.
`kW`, `kWh`	Kilowatt measures power; kilowatt-hour measures energy. The appended hour changes the quantity, not just its scale.
`d`	A conversion day is a fixed 86,400 seconds; a packed calendar date is a separate encoding.
`rad`, `grad`, `turn`	Angle conversion units. Choosing one in CONV does not change the trigonometric angle setting.
N·m, J	Torque and energy share dimensions, but remain separate conversion families.

Bibliography and Further Reading

This bibliography collects the manual's external references and selected further reading. The project source and chapter 15 define SpicyRPN Release 2.0.0's behavior. Other calculators' manuals describe different software or hardware. Online references were checked for this edition on 3 October 2026; access to some journal articles is limited to bibliographic records and abstracts.

SpicyRPN and calculator practice

1. Bryan R. Davies. *SpicyRPN Release 2.0.0: source code and documentation*. Release source ZIP and compiled manual. The application, tests, libraries, licensing notices, and editable manual sources belong together. Use the version-matched source package when comparing implementation details with this manual. Appendix D explains the source layout and build procedure.

2. SpicyRPN. *An RPN Primer*. Online primer; SpicyRPN website. The source of the longer worked expression in Appendix A and the author's account of the calculator's origins in Appendix B. The primer explains notation and the calculator's history; this manual specifies Release 2.0.0's operating rules and supported features.

3. Hewlett-Packard. *HP 12c Platinum Financial Calculator User's Guide*. Edition 5, August 2007. HP part number F2231AA-90001. Official guide (PDF). A companion for financial registers, cash-flow signs, TVM, amortization, bonds, and depreciation. Sections 3–4 and appendices A and E explain financial conventions, stack effects, and formulas. Check this manual for SpicyRPN's entry rules, stack depth, precision, and supported cases.

RPN history and human factors

These sources support the historical discussion and qualified comparison of entry methods in Appendix B. The three calculator-use studies concern particular users, tasks, and devices; their abstracts do not establish a universal advantage for either mode.

4. C. L. Hamblin. "Translation to and from Polish Notation." *The Computer Journal* 5, no. 3 (1962): 210–213. DOI: 10.1093/comjnl/5.3.210. A primary historical reference on translation between expression notations and their use in computation.

5. Hewlett-Packard. *HP Virtual Museum*, calculator collection: 9100A desktop calculator (1968), HP-35 handheld scientific calculator (1972), and HP-12C programmable financial calculator (1981). Manufacturer records for the machines and introduction dates discussed in the historical timeline.

6. Hewlett-Packard. "Reverse Polish Notation." *Hewlett-Packard Journal*, June 1972, sidebar. HP museum scan (PDF). A contemporary explanation of operand-first entry and its connection with a last-in, first-out stack.

7. D. M. Kasprzyk, C. G. Drury, and W. F. Bialas. "Human behaviour and performance in calculator use with Algebraic and Reverse Polish Notation." *Ergonomics* 22, no. 9 (1979): 1011–1019. DOI: 10.1080/00140137908924675; SUNY research record. Comparison of experienced users working with matched calculators; relevant to speed, errors, and the role of prior experience.

8. S. J. Agate and C. G. Drury. "Electronic calculators: which notation is the better?" *Applied Ergonomics* 11, no. 1 (1980): 2–6. DOI: 10.1016/0003-6870(80)90114-3; PubMed record and abstract. A study of calculation performance with algebraic and RPN calculators, including differences associated with users' backgrounds.

9. John G. Kreifeldt. "Hand Calculator Performance Under Interrupted Operation." *Proceedings of the Human Factors Society Annual Meeting* 25, no. 1 (1981): 329–332. DOI: 10.1177/107118138102500187; Publisher abstract. Examines interruption costs and the difficulty of separating notation effects from user selection.

Mathematics, statistics, units, and finance

These further-reading references explain the mathematics behind the calculator. Appendix C gives SpicyRPN's particular formulas, domains, rounding choices, and numerical limitations.

10. National Institute of Standards and Technology. *NIST Digital Library of Mathematical Functions*. Online reference. Chapters 4, 5, and 7 cover elementary functions, the gamma function, and error functions. Consult their definitions and branch conventions when comparing complex logarithms, powers, inverse functions, and gamma-related results.

11. John L. Spouge. "Computation of the Gamma, Digamma, and Trigamma Functions." *SIAM Journal on Numerical Analysis* 31, no. 3 (1994): 931–944. DOI: 10.1137/0731050. The original paper on the approximation named in SpicyRPN's complex gamma implementation. Read alongside the source code for the chosen truncation and precision policy.

12. NIST and SEMATECH. *e-Handbook of Statistical Methods*. Online handbook; Exploratory Data Analysis; Process Modeling. Further reading on descriptive statistics, variation, regression, assumptions, and residual analysis, to help interpret a coefficient or fitted line.

13. Bureau International des Poids et Mesures. *The International System of Units (SI)*. 9th edition, 2019; updated 2026. SI Brochure and current updates. The primary reference for SI definitions, prefixes, and notation. Use alongside the conversion chapter when distinguishing units, quantities, and numerical values.

14. A. Thompson and B. N. Taylor. *Guide for the Use of the International System of Units (SI)*. NIST Special Publication 811, 2008 edition. NIST guide and conversion tables. Appendix B is useful for non-SI conversion factors and conventions. NIST notes that this publication has not been revised for the 2019 SI redefinition; use the SI Brochure above for current SI definitions.

15. Texas Instruments. *BA II Plus Professional Guidebook*. Official guidebook; AOS and chain-calculation setting. Reference for explicit assignment and computation, labeled financial worksheets, and last-answer interaction. SpicyRPN's ALG worksheets adopt these interaction principles while retaining their own expression engine, periodic TVM rate, and existing financial formulas; see sections 4.11 and 4.12.

16. Julie Dahlquist and Rainford Knight. *Principles of Finance*. OpenStax, 2022. Online textbook. Further reading on time value of money, annuities, bond valuation, and investment decisions. Translate the examples into SpicyRPN's signs, periodic rates, and payment timing as described in chapters 11–13.

Numerical libraries, verification, and licensing

17. GNU Project. *GNU MP: The GNU Multiple Precision Arithmetic Library*, version 6.3.0. Project website; Library manual; Copying conditions. Documents the multiple-precision foundation used by MPFR, including build requirements, programming interfaces, and upstream terms.

18. The MPFR team. *GNU MPFR*, version 4.2.2. Project website; Reference manual; Rounding section. The principal upstream reference for SpicyRPN's real-number arithmetic: precision, rounding modes, special values, and function guarantees. Correct rounding of an individual MPFR operation does not make an entire multi-step calculation exact.

19. Andreas Enge, Philippe Théveny, and Paul Zimmermann. *GNU MPC: The GNU Multiple Precision Complex Library*, edition 1.4.1, April 2026. Project website; Reference manual. Describes complex arithmetic over GMP and MPFR, component rounding, signed zeros, and branch cuts. This is the upstream reference for SpicyRPN's complex arithmetic and elementary functions. See sections C.4 and D.5 for the calculator's conventions, composed algorithms, and build setup.

20. The mpmath developers. *mpmath Documentation*, release 1.3.0. Versioned documentation. Documents the independent arbitrary-precision Python library used to generate the project's complex-number test fixtures. Also useful for precision and branch conventions when comparing numerical tools.

21. SQLite project. *Write-Ahead Logging*. SQLite documentation. Explains WAL and shared-memory sidecar files, checkpoints, and persistence behavior. The application source specifies how SpicyRPN configures and uses SQLite.

22. Open Source Initiative. *The MIT License*. License text. Original SpicyRPN code and documentation use this license, reproduced with Bryan R. Davies's copyright notice in section E.1.

23. Free Software Foundation. *GNU General Public License*, version 3, and *GNU Lesser General Public License*, version 3 (2007). GPLv3 text; LGPLv3 text; GNU license collection. The complete texts appear in Appendix E: GPLv3 in section E.2 and LGPLv3 in section E.3. Read these with the source package's `LICENSING.md`, `LICENSE`, and bundled third-party notices. Original SpicyRPN material uses MIT (section E.1); upstream components retain their own license terms and version options.

Building the application and manual

This primary documentation complements Appendix D. Apple's requirements and tool interfaces change; match the documentation to the Xcode and operating-system versions used for a particular build.

24. Apple. *SDK and system requirements*. Xcode documentation. Compatibility tables. Supported host systems, bundled SDKs, deployment targets, simulators, and Swift versions.

25. Apple. *Configuring command-line tools settings*. Apple Developer documentation. Selection of the developer tools used by command-line builds.

26. Apple. *Running your app on simulated or physical devices* and *Enabling Developer Mode on a device*. Running on devices and simulators; Developer Mode. Choosing a run destination and preparing a physical device to run a development build.

27. Apple. *Running tests and interpreting results*. Apple Developer documentation. Test execution, result inspection, and the Xcode workflows used by the project's test scripts.

28. TeX Users Group, MacTeX working group. *MacTeX*. Project website; Download and installation information. The macOS distribution used in the manual's build instructions, including the TeX Live environment and associated tools.

29. LuaTeX development team. *LuaTeX*. Project website and documentation. The typesetting engine underlying this manual's LuaLaTeX build.

30. John Collins. *latexmk*. CTAN package record and documentation. Automated LaTeX builds and the repeated processing needed to resolve contents, references, and other generated material.

31. Markus Kohm. *KOMA-Script*. CTAN package record and manuals. The `scrbook` class and the page-layout, heading, and document-structure facilities used by the manual.

Index

Page numbers link to explanations, worked uses, and reference entries. Commands appear under their legends and descriptive names. Unit abbreviations are collected under *Unit symbols*; ALG labels name displayed functions, not physical-keyboard text commands.

Symbols
- +, addition, 24, 33, 193, 252
- −, subtraction, 24, 33, 193, 252
- ×, multiplication, 24, 33, 193, 252
- ÷, division, 24, 33, 193, 252
- ←, backspace, 28, 37, 193
- $\lceil x \rceil$, ceiling, 75, 195, 251
- $\lfloor x \rfloor$, floor, 75, 195, 251
- π, pi, 81, 197, 252
- +/−, change sign, 22, 31, 199
- =, equals, 31, 199
- Comma (,), 36, 201
- (, opening parenthesis, 33, 207
-), closing parenthesis, 33, 207
- %, percentage key, 26, 36, 78, 207, 252
- ^, power shortcut, 34, 191, 208
- $\bar{x}$, 114, 205, 258, 120
- $\overline{xw}$, 114, 205, 258, 120
- ΔDAYS, 158, 198, 264, 166
- Δ%, 26, 207, 252
- Γ, 80, 201, 252
- $\hat{y}$, 117, 204, 258, 120
- i, imaginary unit, 85, 315
- Σ−, 113, 212, 258, 120
- Σ+, 111, 212, 258, 120

0–9

0 key, 190, 199
10^x, 72, 200, 252
12÷, 125, 214, 259, 137
12×, 125, 214, 259, 137
1 key, 190, 199
$1/x$, 71, 210, 252
2 key, 190, 199
30/360 day count, 158, 198, 264, 166
360-day interest, 155, 202, 262, 166
365-day interest, 155, 202, 262, 166
3 key, 190, 199
4 key, 190, 199
5 key, 190, 199
6 key, 190, 199
7 key, 190, 199
8 key, 190, 199
9 key, 190, 199

A

ABS, 75, 192, 252
`abs` (ALG label), 35, 192, 252
Absolute value, 75, 192, 252
Accepted worksheet inputs, 42
Accessibility, 10
Accumulator recall, 118, 209, 258, 120
Accuracy
- input, 46
- working digits versus correct digits, 55, 250

`acos` (ALG label), 35, 213, 252
`acos[degrees]` (ALG label), 35, 213, 252
`acos[gradians]` (ALG label), 35, 213, 252
acosh, 75, 202, 252
`acosh` (ALG label), 35, 202, 252
`acos[radians]` (ALG label), 35, 213, 252
Acre (acre), 101, 218, 268
Actual days, 158, 198, 264, 166
Adding days, 157, 198, 264, 166
ADD (ALG cash-flow worksheet), 150, 195
ADD SAMPLE (ALG), 120, 42
Adding statistical samples, 111, 212, 258, 120
Addition, 24, 33, 193, 252
Affine conversion, 103, 307
Agate and Drury study, 246, 249, 333
ALG, 8, 171, 211
- angle suffixes in functions, 35, 213, 252
- available functions, 41
- associative memory, 64, 193
- clearing, 40
- complex display and component editing, 31, 87, 196
- continuing a result, 39

displayed function names, 35, 201
editing, 37
errors, 40
expression entry, 31
function arguments, 36
financial assignments and CPT, 42, 137
function preview, 35, 39
implicit multiplication, 34
maximum expression size, 40, 220
mode switching, 41
negative numbers, 34
postfix functions, 35
powers, 34
precedence, 33
prefix functions, 35
replacing a trailing operand, 37, 39
retained expression and worksheets, 42, 173
statistics, 120
storing an operand versus a result, 42
worksheets and multiple outputs, 42, 197
retained angle units, 35, 198, 252
versus RPN, 242, 244
AMORT, 134, 192, 262, 137
Amortization, 134, 192, 262, 137
beginning payments, 134, 262
rounding payments, 138, 262
stack and register effects, 135, 192, 262, 137
Angle units, 73, 198, 252
conversion versus setting, 106
Annual rate
nominal versus effective, 125, 214, 259, 137
simple interest, 155
Annuities
beginning and end payments, 126, 259
present value, 130
savings, 131
Annuity due, 126, 194, 259
Annunciator, 215, 308
ANS, 39, 192
worksheet updates and output selection, 39, 42
Answer, previous, 39, 192
Antilogarithms, 72, 200, 252
APPEND (ALG cash flows), 150, 194
Apple-silicon Mac, 272, 280, 283
Apply button, 216
Arbitrary precision, 47, 308
Arccosine, 73, 213, 252
Arcsine, 73, 213, 252
Arctangent, 73, 213, 252
Argument separator, 36, 201
Arithmetic, 24, 33, 193, 252
Arithmetic mean, 114, 205, 258, 120
Arity, unary and binary functions, 69, 187
arm64, 276
`asin` (ALG label), 35, 213, 252
`asin[degrees]` (ALG label), 35, 213, 252
`asin[gradians]` (ALG label), 35, 213, 252
asinh, 75, 202, 252
`asinh` (ALG label), 35, 202, 252
`asin[radians]` (ALG label), 35, 213, 252
ASSOC flag, 9, 215
Associative memory, 63, 193, 64
numeric equality, 63
real-component keys in both modes, 63, 193, 64
saved duplicate keys, 170
Associativity, 33, 34, 308
atan2, 73, 193, 252
`atan2` (ALG label), 35, 193, 252
`atan2[degrees]` (ALG label), 35, 193, 252
`atan2[gradians]` (ALG label), 35, 193, 252
`atan2[radians]` (ALG label), 35, 193, 252
`atan` (ALG label), 35, 213, 252
`atan[degrees]` (ALG label), 35, 213, 252
`atan[gradians]` (ALG label), 35, 213, 252
atanh, 75, 202, 252
`atanh` (ALG label), 35, 202, 252
`atan[radians]` (ALG label), 35, 213, 252
Automatic saving, 12, 170
Avoirdupois, 101, 308

B

Backspace, 28, 37, 193
Balloon payments, 129, 259
Bar (bar), 104, 218, 268
Base logarithm, 72, 203, 252
BEGIN, 126, 194, 259
BGN, 126, 194, 259

BGN flag, 9, 215
Bibliography, 332
Binary floating-point arithmetic, 46, 250
Binary logarithm, 72, 203, 252
Bisection and bracketing, 263
Bit, 250, 309
Blue shift, 8, 174, 211
Bond price, 159, 160, 208, 265, 166
Bonds
  accrued interest, 159, 160, 265
  clean and dirty price, 159, 265
  coupon dates, 159, 265
  coupon rate, 160, 208, 265, 166
  inputs and conventions, 159
  maturity and settlement, 159, 265
  one remaining coupon, 160, 265
  price/yield example, 160
  redemption at 100, 159, 265
  semiannual convention, 159, 265
Book value, 162, 163, 266
Bounded IRR, 148, 203, 262, 150
Brackets
  ALG postfix notation, 35
  IRR interval, 148, 263
Branch cuts, 94, 254
BTU, 105, 218, 268
Build troubleshooting, 289
Bundle identifier, 279
Burroughs B5000, 247

C

CAL, 52, 194
Calculations
  independent starting state, 14, 177
  preparing a new problem, 177
Calorie, 105, 218, 268
CANCEL, 10, 174, 194, 42
Cancellation, 55
  numerical, 54, 55, 250
  prefixes and shifts, 10, 174, 194, 211, 42
Caret (power shortcut), 191, 208, 252
Cash-flow frequency, 142, 194, 262, 150
Cash-flow index, 145, 310
Cash-flow inspector, 144, 212, 150
Cash flows
  arithmetic storage, 144, 195, 262, 150
  capacity, 152, 220
  clearing, 152
  comparison examples, 151
  equal spacing, 140, 262
  frequency expansion, 142, 194, 262, 150
  indexed storage and recall, 144, 195, 262, 150
  initial investment, 140, 142
  inspection, 144
  keyboard, 141
  NPV equation, 146, 262
  repetitions, 142
  selection, 145
  timing and signs, 140
  zero amounts, 142, 152
Cash-flow signs
  investor and borrower, 125, 140
  TVM, 125
`ceil` (ALG label), 35, 195, 251
Ceiling, 75, 195, 251
Celsius, 103, 218, 268
Centimeter (cm), 101, 218, 268
CF_0 (CF0), 142, 194, 262, 150
CF flag, 9, 215
CF index field, 144, 212, 150
CF_j (CFj), 142, 194, 262, 150
Change sign, 22, 31, 199
Charles Hamblin, 247, 249, 332
CL_{cf} (CLcf), 67, 110, 152, 196
Clean builds, 282
Clean price, 160, 310
CLEAR STATISTICS (ALG), 120, 196
CLEAR, 19, 40, 175, 195
Clear expression, 19, 40, 175, 195
Clearing
  ALG expression, 40, 195
  cash-flow list, 152, 196
  choosing an operation, 175
  financial registers, 128, 196
  Last X and Last Y preserved, 19, 176
  memory-bank groups, 67, 204
  no factory reset, 67, 175
  statistics, 110, 196
  unfinished entry, 175
Clear stack, 19, 40, 175, 195
Clear X, 19, 40, 175, 195
CL_{fin} (CLfin), 67, 110, 152, 196
Close button, 216
Closing parenthesis, 33, 207
CL_{Σ} (CL Sigma), 67, 110, 152, 196
CLx, 19, 40, 175, 195
CMPLX, 84, 92, 196, 254

Code signing, 279
Collapse stack, 17, 213
Combinations, 79, 207, 252
Comma, 36, 201
Command Line Tools, 272, 335
Command reference, 186
Common logarithm, 72, 203, 252
Complementary error function, 81, 200, 252
COMPLEX flag, 9, 215
Complex mode, 84, 92, 196, 254
Complex numbers
  ALG component editing, 87, 196
  angle units, 88, 254
  arithmetic, 90, 254
  components, 84
  conjugate, 90
  entry, 85
  Gamma and log-Gamma, 94, 254
  hidden components, 92
  logarithms in radians, 94, 254
  MPC arithmetic, 90, 96, 254
  negative-real roots, 94, 254
  numerical limits, 96
  one stack position, 13, 84
  polar conversion formulas, 88, 254
  polar form, 88
  principal values, 94
  real-only functions, 93
  rectangular representation, 84, 85
  supported functions, 90
  swapping components, 90
  zero magnitude and phase, 88, 254
Component selection, 85, 196, 254
Compound interest, 125, 259
Conditioning of a calculation, 55, 263
Conjugate, complex, 90
Constants, 81
  precision at creation, 54, 81
CONV, 98, 99, 197, 218, 268
Conversions
  acceleration, 104, 218, 268
  affine factors, 103, 268
  ALG effects, 99
  all unit names, 218
  angle, 106, 218, 268
  area, 101, 218, 268
  controls, 98
  energy, 105, 218, 268
  energy and power, 105
  exact factors, 107, 268
  force, 104, 218, 268
  force and pressure, 104
  frequency, 105, 218, 268
  further reading, 334
  length, 101, 218, 268
  mass, 101, 218, 268
  physical plausibility, 103, 218
  power, 105, 218, 268
  precision, 107
  pressure, 104, 218, 268
  real-component behavior, 93, 99
  rounding, 107
  RPN effects, 99
  same-unit operation, 107, 218
  speed, 104, 218, 268
  speed and acceleration, 104
  temperature, 103, 218, 268
  time, 105, 218, 268
  time and frequency, 105
  torque, 104, 218, 268
  units are not stored tags, 99, 197, 268
  volume, 101, 102, 218, 268
Convert X, 98, 99, 197, 218, 268
Corrections
  RPN, 28
Correlation, 117, 204, 258, 120
COS, 73, 213, 252
`cos` (ALG label), 35, 213, 252
`cos[degrees]` (ALG label), 35, 213, 252
`cos[gradians]` (ALG label), 35, 213, 252
cosh, 75, 202, 252
`cosh` (ALG label), 35, 202, 252
Cosine, 73, 213, 252
COS^{-1}, 73, 213, 252
`cos[radians]` (ALG label), 35, 213, 252
Coupon dates, 159, 265
COMPUTE (ALG worksheets), 42, 197
CPT (ALG computation), 42, 190, 197
CPT flag, 215
Crossover depreciation, 164, 198, 266, 166
Cubic centimeter (cm^3), 101, 218, 268
Cubic foot (ft^3), 101, 218, 268
Cubic inch (in^3), 101, 218, 268
Cubic meter (m^3), 101, 218, 268

D

DATE, 157, 198, 264, 166
Date order, 156, 205, 264
Dates
  calendar range, 156, 264

day counts, 158
entry, 156
fractional encoding positions, 156, 264
Gregorian and leap-year rules, 156, 264
reversed 30/360 intervals, 158, 264
Day difference, 158, 198, 264, 166
Day (d), 105, 218, 268
DB, 164, 198, 266, 166
DB→SL, 164, 198, 266, 166
Decimal places, 46, 48
Decimal point, 22, 31, 199
Declining-balance depreciation, 164, 198, 266, 166
Deepest register, 13, 16
Default settings, 4, 217
DEG, 73, 198, 252
DEG flag, 9, 215
Degree Celsius (°C), 103, 218, 268
Degree Fahrenheit (°F), 103, 218, 268
Degree Rankine (°R), 103, 218, 268
Degree (°), 106, 218, 268
Delete key, 28, 37, 193
ΔDAYS, 158, 198, 264, 166
ΔFIX flag, 215
Δ%, 26, 36, 78, 207, 252
Deployment targets, 272, 335
library settings, 276
Depreciable base, 162, 266
Depreciation
book value versus value above salvage, 162, 266
declining balance, 164
factor in percent, 164, 198, 266, 166
final-year write-off, 164, 166, 266
fractional life, 166
independent year requests, 163, 198, 266, 166
inputs, 162
remaining life, 164, 266
rounding, 166
straight line, 163
straight-line crossover, 164
sum of years' digits, 163
Depth field, 20, 213
DerivedData, 282
Developer Mode, 279, 335
Developer tools, 272
Development team, 279
Device versus Simulator libraries, 276
Digital Library of Mathematical Functions (DLMF), 333
Digit ruler, 52, 194
Digits, 22, 31, 199
DIGITS flag, 9, 215
DIVIDE (ALG cash-flow worksheet), 150, 195
Dimension selector, 98, 197
Dirty price, 160, 312
Discarding X, 18, 199
Display
ALG expression line, 4, 31, 87
complex component rows, 84, 96
digit ruler, 52
exponents, 52
FIX scientific fallback, 48, 200, 250
orientation, 4
pinned exponents, 52
scrolling, 52
status messages, 4, 222
Display digits, 47, 54, 208, 250
Division, 24, 33, 193, 252
DMY, 156, 205, 264
DMY flag, 9, 215
Domain errors, 82, 180, 222
DRG, 73, 198, 252
DROP, 18, 199
Duplicating X, 15, 199

E

e, constant, 81, 197, 252
EEX, 22, 31, 199
Effective annual rate, 125, 214, 259, 137
END, 126, 194, 259
ENG, 48, 50, 200, 250
ENG flag, 9, 215
Engineering notation, 48, 50, 200, 250
English Electric KDF9, 247
ENTER, 15, 199
accepting an ALG worksheet field, 42
committing or duplicating RPN X, 15
Entry
commit versus duplicate, 15, 199
incomplete exponent, 22, 40, 199
mantissa and exponent signs, 22, 31
saved punctuation, 170, 173
ENTRY flag, 9, 215
Entry modes, 8, 171, 211
retained work, 171

Ephemeral calculator state, 283
Equals, 31, 199
erf, 81, 200, 252
`erf` (ALG label), 35, 200, 252
erfc, 81, 200, 252
`erfc` (ALG label), 35, 200, 252
Error function, 81, 200, 252
Error messages, 180, 222
Errors
  calculation recovery, 180
  domain and nonfinite values, 82, 222
  financial solver, 148, 222, 263
  insufficient stack operands, 20, 222
  invalid settings, 47, 222
  prefix routing, 174, 222
  saving and restoring, 182
Escape, 174, 191
Estimated Y, 117, 204, 258, 120
e^x, 72, 200, 252
Euler's number e, 81, 197, 252
Evaluating an expression, 31, 199
Evaluation order
  ALG, 33, 34
  RPN, 228, 229
Exchange X and Y, 18, 213
Exercises
  answers, 30, 236
  RPN practice, 29, 236
`exp` (ALG label), 35, 200, 252
Exponential functions, 72, 200, 252
Exponent range, 55, 250

F

`fact` (ALG label), 35, 200, 252
Factorial, 79, 200, 252
  complex extension, 90, 254
Fahrenheit, 103, 218, 268
f flag, 215
Financial calculations
  additional tools, 154
  further reading, 334
  input occupancy, 62, 222
  Last-register effects, 167, 186
  precision, 138
  real-only inputs, 93, 220
  register reuse, 128, 167
  retained state, 167
  solver tolerances, 138, 263
Financial keys
  explicit ALG computation, 42, 137
  solve after explicit store, 127, 206, 259, 137
  solving after recall, 127, 206, 259, 137
  RPN store-or-solve rule, 127
  unwritten inputs are zero, 127, 206, 259, 137
Financial registers, 124, 127, 206, 259, 327, 137
  clearing, 128
FIN flag, 9, 215
FIX, 48, 50, 200, 250
  ALG sequence, 50
  RPN sequence, 50
Fixed notation, 48, 50, 200, 250
Fixed stack, 20, 213
FIX flag, 9, 215
Flags, 9
  ASSOC, 215
  BGN, 215
  CF, 215
  COMPLEX, 215
  CPT, 215
  DEG, 215
  ΔFIX, 215
  DIGITS, 215
  DMY, 215
  ENG, 215
  ENTRY, 215
  f, 215
  FIN, 215
  FIX, 215
  FUNC, 215
  g, 215
  GRAD, 215
  HYP, 215
  INVHYP, 215
  MCLR, 215
  MDY, 215
  MEM, 215
  RAD, 215
  RCL, 215
  SCI, 215
  STAT, 215
  STO, 215
  STO+, 215
  STO−, 215
  STO÷, 215
  STO×, 215
  TURN, 215
Floor, 75, 195, 251

`floor` (ALG label), 35, 195, 251
Fluid ounce, 102, 218, 268
Focus numeric entry, 10, 191
Foot per second squared (ft/s^2), 104, 218, 268
Foot per second (ft/s), 104, 218, 268
Foot (ft), 101, 218, 268
Forty-two example, 231
FRAC, 75, 202, 251
`frac` (ALG label), 35, 202, 251
Fractional part, 75, 202, 251
Fractional periods, 132
  HP odd-period difference, 132, 259
From and To units, 98, 197
FSE, 48, 50, 200, 250
f shift, 8, 174, 211
Full-stack inspector, 17, 213
FUNC, 36, 201
FUNC flag, 9, 215
Function arguments, 36, 201
Function domains, 82, 220
Further reading, 332
Future value, 131
FV, 124, 127, 206, 259, 327, 137

G

Gallon, 102, 218, 268
Γ (Gamma), 80, 201, 252
`gamma` (ALG label), 35, 201, 252
Gamma function, 80, 201, 252
  further reading, 333
  poles and signs, 80, 252
g flag, 215
Gigahertz (GHz), 105, 218, 268
Glossary, 307
GMP, 275, 334
GOAT, 2, 201
Goat sound, 2, 201
GPL and LGPL, 291, 335
  GNU General Public License, version 3, 292
  GNU Lesser General Public License, version 3, 304
GRAD, 73, 198, 252
GRAD flag, 9, 215
Gradian (grad), 106, 218, 268
Gram (g), 101, 218, 268
Group clear, 67, 175, 204
g shift, 8, 174, 211
Guard digits, 55, 254

H

Hamblin, C. L., 332
Header and linker errors, 289
Hectare (ha), 101, 218, 268
HELP, 10, 201
Hertz (Hz), 105, 218, 268
Hidden imaginary components, 92
Horner evaluation, cash flows, 262
Horsepower, 105, 218, 268
Hour (h), 105, 218, 268
HP-12C, 247
HP 12c Platinum guide, 332
HP-35, 247
HP 9100A, 247
HP calculators
  four-register stack comparison, 21
  HP 12c financial conventions, 127, 132, 259, 332
  HP 35 and HP 9100A, 247, 333
HP Virtual Museum, 333
HYP, 75, 202, 252
Hyperbolic cosine, 75, 202, 252
Hyperbolic functions, 75, 202, 252
Hyperbolic sine, 75, 202, 252
Hyperbolic tangent, 75, 202, 252
HYP flag, 9, 215
HYP^{-1}, 75, 202, 252

I

IEEE remainder, 77, 205, 251
Imaginary unit, 85, 315
Immutable values, 54, 315
Imperial fluid ounce (Imp fl oz), 102, 218, 268
Imperial gallon (Imp gal), 102, 218, 268
Imperial long ton (long ton), 101, 218, 268
Imperial pint (Imp pt), 102, 218, 268
Imperial quart (Imp qt), 102, 218, 268
Imperial units, 102
Implicit multiplication, 34
Inch (in), 101, 218, 268
Independent expected results, 287
Indexed root, 34, 71, 208, 252
Infinity, 55, 82, 222
INFIX, 47, 54, 208, 250
Infix notation, 33, 226, 242
Initial cash flow, 142, 194, 262, 150
Installation, 3
INT, 75
  integer part, 75, 202, 251, 315

simple interest, 155, 202, 262, 321, 166
Integer part, 75, 202, 251
Intercept, 116, 204, 258, 120
Intel Mac, 272, 280, 283
Interest
compound versus simple, 125, 155
negative rates, 131, 259
periodic rates, 125
TVM solve, 131
Internal rate of return, 147, 203, 262, 150
International System of Units (SI), 334
International Table BTU (BTU), 105, 218, 268
`inv` (ALG label), 35, 210, 252
Inverse hyperbolic cosine, 75, 202, 252
Inverse hyperbolic functions, 75, 202, 252
Inverse hyperbolic sine, 75, 202, 252
Inverse hyperbolic tangent, 75, 202, 252
Inverse trigonometric functions, 73, 213, 252
INVHYP flag, 9, 215
iPhone and iPad builds, 281
i register, 124, 127, 206, 259, 327, 137
IRR, 147, 203, 262, 150
bounds, 148, 203, 262, 150
choosing a root, 148, 262
does not store i, 147, 203, 262, 150
initial guess, 147
multiple roots, 148, 203, 262, 150
no sign change, 148, 263
numerical limits, 152
versus NPV, 151
IRR[a,b], 148, 203, 262, 150
Isolated test database, 283
Iteration limits, 220, 263

J

Jan Lukasiewicz, 247
Joule (J), 105, 218, 268

K

k, index or unit prefix, 324
Kasprzyk, Drury, and Bialas study, 246, 249, 333
Kelvin (K), 103, 218, 268
Key-click feedback, 7
Keyboard
positions in RPN and ALG, 6, 188
ALG map, 190
click sound, 7
entry and arithmetic map, 190
Financial panel map, 189
panels, 6
physical, 10
RPN middle rows, 189
Scientific panel map, 188
unassigned shifts, 188, 222
Keyboard shortcuts, 10, 191
Backspace, Delete, 191, 193
), 191, 207
„ 191, 201
., 191, 199
0–9, 191, 199
/ or ÷, 191, 193, 252
e, E, 191, 199
Escape, 191, 211
f, F, 191, 211
g, G, 191, 211
- or minus sign, 191, 193, 252
* or ×, 191, 193, 252
(, 191, 207
p, P (pi), 191, 197, 252
+, 191, 193, 252
^, 191, 208, 252
Return, newline, =, 191, 199
s, S, 191, 199
Key guide, 10, 201
Keystroke counts, 244
Kilogram (kg), 101, 218, 268
Kilohertz (kHz), 105, 218, 268
Kilojoule (kJ), 105, 218, 268
Kilometer per hour (km/h), 104, 218, 268
Kilometer (km), 101, 218, 268
Kilonewton (kN), 104, 218, 268
Kilopascal (kPa), 104, 218, 268
Kilowatt-hour (kWh), 105, 218, 268
Kilowatt (kW), 105, 218, 268
Knot (kn), 104, 218, 268
KOMA-Script, 336
Kreifeldt interruption study, 246, 249, 333

L

Landscape layout, 6, 188
LAST X, 19, 203
ordinary versus financial recall, 65, 209
preserving operands during correction, 28, 203
LAST Y, 19, 203
unlimited single-value stack, 19, 187

LaTeX manual, 288
latexmk, 288, 336
Learning RPN, 226, 245
`lgamma` (ALG label), 35, 201, 252
LGPL, 304
Library archive checksums, 276
License and redistribution, 274, 291, 335
Linear estimation, 117, 204, 258, 120
Linear regression, 116, 204, 258, 120
Liter (L), 101, 218, 268
LN, 72, 203, 252
$\ln 2$, constant, 81, 197, 252
`ln` (ALG label), 35, 203, 252
`lnfact` (ALG label), 35, 200, 252
$\ln \Gamma$, 80, 201, 252
$\ln x!$, 80, 200, 252
Loan balance
    amortization, 134, 192, 262, 137
Loans
    balloon or residual amount, 129
    number of payments, 132
    payments, 129
LOG, 72, 203, 252
`log10` (ALG label), 35, 203, 252
LOG_2 (LOG2), 72, 203, 252
`log2` (ALG label), 35, 203, 252
Logarithmic factorial, 80, 200, 252
Logarithmic Gamma, 80, 201, 252
Logarithms, 72, 203, 252
`logbase` (ALG label), 35, 203, 252
$\log_y x$, 72, 203, 252
Long ton, 102, 218, 268
Lower and Upper IRR bounds, 148, 203, 150
L.R., 116, 204, 258, 120
LuaLaTeX, 288, 336
LuaTeX, 336

M

Mac build, 280
MacTeX, 288, 336
Magnitude, 88
    ABS command, 75, 192, 252
    complex value, 88, 254
Mantissa, 48, 316
Manual
    chapter ownership, 288
Manual build script, 288
MathBridge, 275
Mathematical formulas, 250
Mathematical libraries
    building, 276
    manuals, 334
Mathematical symbols, 324
    arithmetic and precision, 325
    cash flows and solvers, 328
    complex numbers, 326
    dates and bonds, 329
    depreciation, 330
    entry and overloaded letters, 324
    financial registers, 327
    statistical accumulators, 112, 258
    statistics, 327
    unit symbols and prefixes, 330
Math-library path errors, 289
MCLR, 67, 175, 204
    canceling, 10, 174, 194, 42
MCLR flag, 9, 215
MDY, 156, 205, 264
MDY flag, 9, 215
Mean, 114, 205, 258, 120
Mechanical horsepower (hp), 105, 218, 268
Megahertz (MHz), 105, 218, 268
Megajoule (MJ), 105, 218, 268
Megapascal (MPa), 104, 218, 268
Megawatt (MW), 105, 218, 268
MEM flag, 9, 215
Memory
    arithmetic storage, 60
    associative, 63, 193, 64
    banks, 57
    clearing, 67, 175, 204
    complex values, 62, 92
    financial registers, 66
    full stored precision, 54, 61
    occupancy indicators, 57
    ordinary registers, 58
    real-mode stores and recalls, 62, 92
    reusing constants, 61
    RPN planning, 235
    shared between modes, 65
    statistical registers, 66
    transfer between modes, 41, 65
    unset registers, 62
    zero does not clear occupancy, 62, 215
Meter per second squared ($\mathrm{m/s^2}$), 104, 218, 268
Meter per second (m/s), 104, 218, 268
Meter (m), 101, 218, 268
Metric tonne (t), 101, 218, 268

Micrometer (μm), 101, 218, 268
Microsecond (μs), 105, 218, 268
Mile per hour (mph), 104, 218, 268
Mile (mi), 101, 218, 268
Millibar (mbar), 104, 218, 268
Milligram (mg), 101, 218, 268
Milliliter (mL), 101, 218, 268
Millimeter (mm), 101, 218, 268
Millisecond (ms), 105, 218, 268
Minute (min), 105, 218, 268
MIT License, 291, 335
MOD, 77, 205, 251
`mod` (ALG label), 35, 205, 251
Mode switching, 8, 171, 211
Monthly shortcuts, 125, 214, 259, 137
MPC, 254, 96, 278, 317, 334
MPFR, 275, 334
mpmath, 287, 335
Multiple precision, 3
Multiplication, 24, 33, 193, 252
MULTIPLY (ALG cash-flow worksheet), 150, 195

N

NaN (not a number), 55, 82, 222
Nanometer (nm), 101, 218, 268
Nanosecond (ns), 105, 218, 268
Native macOS application, 280
Natural logarithm, 72, 203, 252
Nautical mile (nmi), 101, 218, 268
nCr, 79, 207, 252
`nCr` (ALG label), 35, 207, 252
Nearest integer, 53, 75, 206, 251
`negate` (ALG label), 35, 199
Negative numbers, 22, 34
Negative powers and roots, 71, 94
Net present value, 145, 146, 206, 262, 150
Newton-meter (N·m), 104, 218, 268
Newton (N), 104, 218, 268
Next cash flow, 142, 194, 262, 150
NINT, 53, 75, 206, 251
NIST SP 811, 334
NIST statistical handbook, 334
N_j (Nj), 142, 194, 262, 150
Nominal annual rate, 125, 214, 259, 137
Nonfinite results, 82, 222
nPr, 79, 207, 252
`nPr` (ALG label), 35, 207, 252
NPV, 145, 146, 206, 262, 150
  discount equation, 146, 262
  time-zero flow, 140, 262
  versus IRR, 151
n register, 124, 127, 206, 259, 327, 137
  cash-flow index, 145
n, sample count, 112, 118, 209, 258, 120
Number entry, 22, 31, 199
Numerical limits, 55, 220, 270
Numeric literal, 22, 317
Numeric memory keys, 63, 193, 64

O

Obtaining the source, 274
Opening parenthesis, 33, 207
Operand order, 16, 37, 234
Operation selector (ALG), 42
Operand recovery, 19, 203
Orange shift, 8, 174, 211
Ordinary annuity, 126, 194, 259
Ordinary registers, 58, 60, 62, 209
Ounce (oz), 101, 218, 268
Overflow and underflow, 55, 250

P

Packed dates, 156, 205, 264
Panel paging, 6, 188
Parentheses, 33, 207
  RPN interpretation, 228
Pascal (Pa), 104, 218, 268
Payment timing, 126, 194, 259
`pctchange` (ALG label), 35, 207, 252
`pcttotal` (ALG label), 35, 207, 252
PDF manual
  building, 288
Pending commands, 9
  canceling, 8, 174, 211
Percent (%), 26, 36, 78, 207, 252
Percentage change, 26, 36, 78, 207, 252
Percentage of total, 26, 36, 78, 207, 252
Percentages, 78
  ALG binary functions, 36, 207, 252
  discounts and markups, 26
  first calculations, 26
  preserving Y in RPN, 26, 78
  textual percent is remainder, 77, 205, 251
`percent` (ALG label), 35, 207, 252
Periods
  cash-flow cutoff n, 145
  solved whole-number convention, 132, 259
  TVM rounding, 132

Permutations, 79, 207, 252
Persistence, 170, 173, 318
Phase, 88
Physical-device build, 281, 335
Pi (π), 81, 197, 252
Pint, 102, 218, 268
PMT, 124, 127, 206, 259, 327, 137
Polar angle, 88, 254
Polar/rectangular toggle, 88, 196, 254
Polish notation, 247, 332
Population standard deviation, 116
Portrait layout, 6, 188
Postfix functions, 35, 226
Postfix notation, 226, 247
Pound-foot (lbf·ft), 104, 218, 268
Pound-force foot, 104, 218, 268
Pound-force (lbf), 104, 218, 268
Pound per square inch (psi), 104, 218, 268
Pound (lb), 101, 218, 268
`pow10` (ALG label), 35, 200, 252
`power[2]` (ALG label), 35, 212, 252
Powers, 34, 71, 208, 252
  first calculations, 24
PRC, 159, 160, 208, 265, 166
PREC, 47, 54, 208, 250
Precedence, 33, 242
Precision
  choosing, 55
  decimal digits to MPFR bits, 46, 250
  display, 46
  existing values, 54
  iterative financial results, 138, 263
  recomputing old values, 54
  working, 46
Precision settings, 47, 54, 208, 250
Prefix functions, 35, 36
Present value, 130
Previous answer, 39, 192
Primary result (ALG), 39, 42, 319
Principal values, 94, 254
Principles of Finance, 334
Problem reports, 184
Project architecture, 275
Proleptic Gregorian calendar, 156, 264
P↔R, 88, 196, 254
PV, 124, 127, 206, 259, 327, 137

Q

Quadrant-aware angle, 73, 193, 252
Quart, 102, 218, 268

R

R0, 58, 209
R0–R9 ordinary registers, 57, 58, 209
R1, 58, 209
R2, 58, 209
R3, 58, 209
R4, 58, 209
  sample count, 112, 118, 209, 258, 120
R4–R9 statistical copies, 118, 209, 258, 120
R5, 58, 209
  sum of x, 112, 118, 209, 258, 120
R6, 58, 209
  sum of x squared, 112, 118, 209, 258, 120
R7, 58, 209
  sum of y or weights, 112, 118, 209, 258, 120
R8, 58, 209
  sum of y squared, 112, 118, 209, 258, 120
R9, 58, 209
  sum of paired products, 112, 118, 209, 258, 120
RAD, 73, 198, 252
RAD flag, 9, 215
Radian (rad), 106, 218, 268
Random numbers, 81, 209, 252
Rankine, 103, 218, 268
RAN#, 81, 209, 252
RECALL (ALG worksheets), 64, 150
RECALL SUMMATIONS (ALG), 120, 209
RCL, 58, 60, 62, 209
  financial registers, 127, 137
  indexed cash flows, 144, 195, 262, 150
RCL flag, 9, 215
RCL_{Σ} (RCL Sigma), 118, 209, 258, 120
RCL_{x} (RCLx), 63, 193, 64
  ALG worksheet, 64, 190
r, correlation, 117, 204, 258, 120
R↓, 18, 210
Real/imaginary swap, 90, 196, 254
Real mode, 92, 216
Real portion only warning, 93, 222
Rebuilding modified source, 282
Recall, 58, 60, 62, 209
Recall into X (RPN), 144, 212, 150
Reciprocal, 71, 210, 252
Re-entry after errors, 180

Reference fixtures, 287
Regression
correlation restrictions, 117, 258
horizontal and vertical lines, 116, 117
least squares, 116, 258
Re/Im, 85, 87, 196, 254
Relaunch and restoration, 170, 173
REMOVE SAMPLE (ALG), 120, 212
REM, 77, 205, 251
`remainder` (ALG label), 35, 205, 251
Remainders, 77, 205, 251
Removing statistical samples, 113, 212, 258, 120
REPEAT (ALG cash flows), 150, 194
Repeated equals, 31, 199
Residual, 116, 263, 320
Residual values, 129, 130
Re↔Im, 90, 196, 254
Return key, 191, 199
RND, 53, 138, 210, 251, 137
financial workflow, 138
Roll down, 18, 210
Roll up, 18, 210
`root` (ALG label), 35, 208, 252
Roots, 71
`round` (ALG label), 35, 206, 251
Rounding
away from zero, 53, 252
display, 53
financial period count, 132, 259
nearest, ties even, 53, 251
precision versus exponent, 250
stored value, 53, 138, 210, 251, 137
RPN, 8, 171, 211
exercises, 236
extended example, 231
first calculations, 22
history, 247, 333
keystroke comparison, 244
learning curve, 245
notation, 226
practice answers, 236
preparing a known starting state, 14
research evidence, 246, 333
stack traces, 228
translating expressions, 228
working with intermediate results, 25, 241
RPN Primer, 231, 248, 332
R↑, 18, 210

S

S0 and S1, current RPN X and Y, 13, 17, 186, 324
Salvage value, 162, 266
Sample standard deviation, 116, 211, 258, 120
Saved operands, 19, 203
Saved work, 12
automatic saving, 170
failed save or restore, 182
independent workspaces, 171
prefixes and tool fields, 170, 173
resuming entry, 173
Savings, 131
SCI, 48, 50, 200, 250
Scientific functions
availability in ALG, 41, 69
domains, 82
further reading, 333
keyboard, 69
Scientific notation, 48, 50, 200, 250
SCI flag, 9, 215
Scrolling
digits, 52
stack registers, 17
SD, 116, 211, 258, 120
SDK errors, 289
SDK selection, 276, 335
Secant method, 263
Second (s), 105, 218, 268
Shift keys, 8, 174, 211
Short ton, 102, 218, 268
SHOW, 144, 212, 150
Show financial panel, 6, 10
Show full stack, 17, 213
Show scientific panel, 6, 10
SI Brochure, 334
Σ−, 113, 212, 258, 120
Σ+, 111, 212, 258, 120
Sign convention
finance, 125, 140
Significand, 250, 321
Significant digits, 46, 48
Signing errors, 289
Simple interest, 155, 202, 262, 166
Simulator build, 281, 335
SIN, 73, 213, 252
`sin` (ALG label), 35, 213, 252
`sin[degrees]` (ALG label), 35, 213, 252
Sine, 73, 213, 252

`sin[gradians]` (ALG label), 35, 213, 252
sinh, 75, 202, 252
`sinh` (ALG label), 35, 202, 252
SIN^{-1}, 73, 213, 252
`sin[radians]` (ALG label), 35, 213, 252
SL, 163, 198, 266, 166
Slope, 116, 204, 258, 120
Sound
    GOAT, 201
    key clicks, 7
Source code, 274, 332
Source revision, 274, 332
Source tree, 275
SpicyRPN
    capabilities, 2
    origin, 248, 332
SpicyRPN scheme, 280
Spouge approximation, 254, 333
SQLite, 275, 335
`sqrt` (ALG label), 35, 212, 252
$\sqrt{x}$, 71, 212, 252
Square, 71, 212, 252
Square centimeter (cm^2), 101, 218, 268
Square foot (ft^2), 101, 218, 268
Square inch (in^2), 101, 218, 268
Square kilometer (km^2), 101, 218, 268
Square meter (m^2), 101, 218, 268
Square millimeter (mm^2), 101, 218, 268
Square root, 71, 212, 252
Square yard (yd^2), 101, 218, 268
Stack
    deepest-register repetition, 16
    depth, 20, 213
    discarding deepest values, 16, 20
    drop, 16
    fixed versus unlimited, 20
    inspection, 17, 213
    intermediate results, 25, 228
    left-to-right operand order, 16, 234
    lift, 16
    number entry, 15
    operand order, 16
    planning capacity, 229
    planning long expressions, 229, 231
    registers, 13
    repeated constants, 233
    roll, 18
    unlimited, 20
    zero as an ordinary value, 20
STACK, 20, 213
Standard atmosphere (atm), 104, 218, 268
Standard deviation
    centered sums, 116, 258
    population, 116
    sample, 116, 211, 258, 120
Standard gravity (g_0), 104, 218, 268
STAT (ALG worksheets), 120, 42, 190, 197
STAT flag, 9, 215
Static libraries, 276
Statistics
    accumulators, 112
    clearing, 110
    correcting a pair, 113
    errors, 121
    extrapolation, 117
    further reading, 334
    inspection, 118, 209, 258, 120
    loss of significance, 121, 258
    numerical precision, 121
    observations, 111
    preparation, 110
    n, 112, 118, 209, 258, 120
    saved data, 112
    single-variable samples, 111
    stack effects, 121
    $\sum xy$, 112, 118, 209, 258, 120
    $\sum x$, 112, 118, 209, 258, 120
    $\sum x^2$, 112, 118, 209, 258, 120
    $\sum y$, 112, 118, 209, 258, 120
    $\sum y^2$, 112, 118, 209, 258, 120
    weights versus frequencies, 114, 116
    zero variance, 116, 117, 222
STO, 58, 60, 62, 209
    financial registers, 127, 137
    indexed cash flows, 144, 195, 262, 150
STO×, 60, 127, 144, 209
STO+, 60, 127, 144, 209
STO−, 60, 127, 144, 209
STO÷, 60, 127, 144, 209
STO÷ flag, 215
STO flag, 9, 215
STO+ flag, 215
STO− flag, 215
Stone (st), 101, 218, 268
Store, 58, 60, 62, 209
STORE (ALG worksheets), 64, 150
STO× flag, 215
STO_{xy} (STOxy), 63, 193, 64
    ALG worksheet, 64, 190

Straight-line depreciation, 163, 198, 266, 166
SUBTRACT (ALG cash-flow worksheet), 150, 195
Subtracting days, 157, 198, 264, 166
Subtraction, 24, 33, 193, 252
Sum of years' digits, 163, 198, 266, 166
SWAP, 18, 213
  operand order, 234
SwiftUI, 275
s_x, sample deviation, 116, 211, 258, 120
$\sum x$, sum of x, 112, 118, 209, 258, 120
$\sum x^2$, sum of x squared, 112, 118, 209, 258, 120
$\sum xy$, sum of paired products, 112, 118, 209, 258, 120
s_y, sample deviation, 116, 211, 258, 120
SYD, 163, 198, 266, 166
$\sum y$, sum of y or weights, 112, 118, 209, 258, 120
$\sum y^2$, sum of y squared, 112, 118, 209, 258, 120

T

TAN, 73, 213, 252
`tan` (ALG label), 35, 213, 252
`tan[degrees]` (ALG label), 35, 213, 252
Tangent, 73, 213, 252
`tan[gradians]` (ALG label), 35, 213, 252
tanh, 75, 202, 252
`tanh` (ALG label), 35, 202, 252
TAN^{-1}, 73, 213, 252
`tan[radians]` (ALG label), 35, 213, 252
Temperature
  absolute zero, 103, 218
  offsets, 103
  readings versus intervals, 103, 268
Test oracles, 287, 335
Tests
  Cmd-U, 283
  unit and UI suites, 283, 336
Thermochemical calorie (cal), 105, 218, 268
Thermochemical kilocalorie (kcal), 105, 218, 268
Third-party dependencies, 275
Time value of money, 124, 127, 206, 259, 137
  complete equation, 259
  preparing all inputs, 128
  solving interest, 131
  zero-rate case, 259
Tolerance, 263, 322
Tools
  Apply versus Close, 216
  separate typed fields, 216
Torr (Torr), 104, 218, 268
T%, 26, 36, 78, 207, 252
Trigonometric functions, 73, 213, 252
`trunc` (ALG label), 35, 202, 251
Truncating remainder, 77, 205, 251
Truncation, 75, 202, 251
TURN flag, 9, 215
Turn (turn), 106, 218, 268
Two-coordinate arctangent, 73, 193, 252

U

Undo limitations, 19, 175
Unexpected results, 178
Unfinished work, 173
Unit aliases, 218, 323
Unit conversion, 98, 99, 197, 218, 268
Unit prefixes, 330
Units
  names and abbreviations, 218
Use value (ALG), 42, 197
Unit symbols
  ° — degree, 106, 218, 268
  acre — acre, 101, 218, 268
  atm — standard atmosphere, 104, 218, 268
  bar — bar, 104, 218, 268
  BTU — International Table BTU, 105, 218, 268
  °C — degree Celsius, 103, 218, 268
  cal — thermochemical calorie, 105, 218, 268
  cm — centimeter, 101, 218, 268
  cm^2 — square centimeter, 101, 218, 268
  cm^3 — cubic centimeter, 101, 218, 268
  d — day, 105, 218, 268
  °F — degree Fahrenheit, 103, 218, 268
  ft — foot, 101, 218, 268
  ft^2 — square foot, 101, 218, 268
  ft^3 — cubic foot, 101, 218, 268
  ft/s — foot per second, 104, 218, 268
  ft/s^2 — foot per second squared, 104, 218, 268

g — gram, 101, 218, 268
g_0 — standard gravity, 104, 218, 268
GHz — gigahertz, 105, 218, 268
grad — gradian, 106, 218, 268
h — hour, 105, 218, 268
ha — hectare, 101, 218, 268
hp — mechanical horsepower, 105, 218, 268
Hz — hertz, 105, 218, 268
Imp fl oz — Imperial fluid ounce, 102, 218, 268
Imp gal — Imperial gallon, 102, 218, 268
Imp pt — Imperial pint, 102, 218, 268
Imp qt — Imperial quart, 102, 218, 268
in — inch, 101, 218, 268
in^2 — square inch, 101, 218, 268
in^3 — cubic inch, 101, 218, 268
J — joule, 105, 218, 268
K — kelvin, 103, 218, 268
kcal — thermochemical kilocalorie, 105, 218, 268
kg — kilogram, 101, 218, 268
kHz — kilohertz, 105, 218, 268
kJ — kilojoule, 105, 218, 268
km — kilometer, 101, 218, 268
km^2 — square kilometer, 101, 218, 268
km/h — kilometer per hour, 104, 218, 268
kn — knot, 104, 218, 268
kN — kilonewton, 104, 218, 268
kPa — kilopascal, 104, 218, 268
kW — kilowatt, 105, 218, 268
kWh — kilowatt-hour, 105, 218, 268
L — liter, 101, 218, 268
lb — pound, 101, 218, 268
lbf — pound-force, 104, 218, 268
lbf·ft — pound-foot, 104, 218, 268
long ton — Imperial long ton, 101, 218, 268
m — meter, 101, 218, 268
m^2 — square meter, 101, 218, 268
m^3 — cubic meter, 101, 218, 268
mbar — millibar, 104, 218, 268
mg — milligram, 101, 218, 268
MHz — megahertz, 105, 218, 268
mi — mile, 101, 218, 268
min — minute, 105, 218, 268
MJ — megajoule, 105, 218, 268
mL — milliliter, 101, 218, 268
mm — millimeter, 101, 218, 268
mm^2 — square millimeter, 101, 218, 268
MPa — megapascal, 104, 218, 268
mph — mile per hour, 104, 218, 268
m/s — meter per second, 104, 218, 268
ms — millisecond, 105, 218, 268
m/s^2 — meter per second squared, 104, 218, 268
MW — megawatt, 105, 218, 268
N — newton, 104, 218, 268
nm — nanometer, 101, 218, 268
N·m — newton-meter, 104, 218, 268
nmi — nautical mile, 101, 218, 268
ns — nanosecond, 105, 218, 268
oz — ounce, 101, 218, 268
Pa — pascal, 104, 218, 268
psi — pound per square inch, 104, 218, 268
°R — degree Rankine, 103, 218, 268
rad — radian, 106, 218, 268
s — second, 105, 218, 268
short ton — US short ton, 101, 218, 268
st — stone, 101, 218, 268
t — metric tonne, 101, 218, 268
Torr — torr, 104, 218, 268
turn — turn, 106, 218, 268
US fl oz — US fluid ounce, 102, 218, 268
US gal — US gallon, 102, 218, 268
US pt — US pint, 102, 218, 268
US qt — US quart, 102, 218, 268
W — watt, 105, 218, 268
Wh — watt-hour, 105, 218, 268
yd — yard, 101, 218, 268
yd^2 — square yard, 101, 218, 268
μm — micrometer, 101, 218, 268
μs — microsecond, 105, 218, 268
Universal Mac application, 276, 280, 283
Unlimited stack, 20, 213
Useful life, 166, 323
US fluid ounce (US fl oz), 102, 218, 268
US gallon (US gal), 102, 218, 268
US pint (US pt), 102, 218, 268
US quart (US qt), 102, 218, 268
US short ton (short ton), 101, 218, 268

US units, 102

V

Value provenance, 170, 275
Variance, 116, 323
Version and build identification, 3, 184

W

Watt-hour (Wh), 105, 218, 268
Watt (W), 105, 218, 268
Worksheets (ALG)
- action buttons, 42, 197
- changing the selected operation, 42
- accepted inputs and drafts, 42
- associative memory, 64
- cash flows, 150
- cancellation and relaunch, 42, 174
- financial and date operations, 166, 137
- primary result and ANS, 39, 42
- statistics, 120
- Use value, 42

Weighted mean, 114, 205, 258, 120
Working digits, 47, 54, 208, 250
Write-ahead logging (WAL), 182, 335

X

X and Y
- current stack operands, visible S0 and S1, 13, 16, 186, 324
- statistical coordinates, 111, 258

$\bar{x}$, mean, 114, 205, 258, 120
Xcode, 272, 335
xcodebuild, 280
$x!$, 79, 200, 252
x^2, 71, 212, 252
$x \leftrightarrow y$, 18, 213
$\sqrt[x]{y}$, 34, 71, 208, 252
$\overline{xw}$, weighted mean, 114, 205, 258, 120

Y

Yard (yd), 101, 218, 268
$\bar{y}$, mean, 114, 205, 258, 120
$\hat{y}$, estimate, 117, 204, 258, 120
Yield to maturity, 159, 160, 208, 265, 166
y^x, 34, 71, 208, 252
YTM, 159, 160, 208, 265, 166

Z

Zero
- empty bank versus stored zero, 62, 215
- phase of complex zero, 254
- underflow and displayed zero, 48, 55