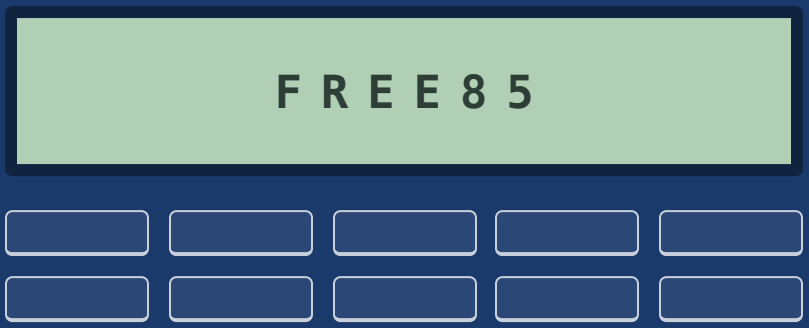


SCIENTIFIC GRAPHING CALCULATOR

The Free85 Guidebook

For firmware 3.0 · Second Edition

chriswilson2020.github.io/Free85

The Free85 Guidebook

The Free85 Guidebook is the complete reference for Free85, an open-source scientific graphing calculator written as original Z80 firmware and running on a faithful emulation of a TI-85-compatible pocket calculator. It is the companion to the Free85 Getting Started Manual: where the manual gets you started, this book covers each subject in depth, one chapter at a time.

Clean-room and licensing notice

Free85 is clean-room software. Everything in it (the firmware, the font, the screen artwork, the tests, the manual, and this book) was written from scratch for this project. It contains no Texas Instruments ROM code, disassembly, fonts, artwork, or binary tables. The TI-85 is referenced only to describe the hardware profile the firmware runs on; the project is not affiliated with or endorsed by Texas Instruments. Because the internals are original, Free85 makes no promise of compatibility with TI programs, files, tokens, ROM calls, or internal data structures. What it does promise is that every physical key, every shifted function printed above a key, and every reachable menu leads to a real, working feature.

Free85 is open source under the MIT License. See the `LICENSE` file for the licence text and `NOTICE.md` for the project notices.

How to read this book

- **Keys** appear in brackets using their keycap labels: [ENTER], [2nd], [ALPHA], [F1] through [F5], [MORE], [x²], and the cursor keys [▲] [▼] [◀] [▶]. A sequence such as [2nd] [ENTER] means press and release [2nd], then press [ENTER]. A bracketed letter such as [H] means the key carrying that letter legend.
- **On-screen text and typed expressions** appear in code spans: the status line shows `RAD AUTO`, and typing [2] [+] [3] puts `2+3` on the entry line. Code spans are also used for command and mode names, including other calculators' names for them.

- **Screenshots** are exact captures of the emulated 128 by 64 pixel LCD. Every image in this book is generated by booting a fresh machine, pressing the listed keys, and photographing the result, so what you see is what the calculator actually shows.
- **Planned-work callouts** flag features that are reachable in a chapter's subject area but not implemented yet. Free85 2.10 completes almost all of the work the first edition flagged this way, and the convention is kept. A callout that survives marks either work that a release deliberately leaves open or a boundary that waits on physical hardware. The work-package number it cites is recorded in appendix D, which reports feature status across the release. They always look like this:

  > **PLANNED**
  > what is missing (Free85 2.0, work package id).

- **Hardware callouts** mark the boundary between emulator and cable:

  > **HARDWARE**
  > what works in the emulator; physical hardware validation is reported separately.

Contents

CHAPTERS

APPENDICES

CHAPTER

1

Operating the Calculator

This chapter covers the everyday mechanics of the calculator: switching it on and off, typing and editing on the home screen, recalling previous work, moving through menus, adjusting the system modes, finding functions in the catalog, and reading the error screen. Everything else in this book builds on the habits formed here.

Power and boot

1-1

Press ON and the calculator boots straight to the home screen:

The home screen after switching on

Reading from the top: the status line shows the angle mode and display format on the left (`RAD AUTO` after a fresh boot) and the editor state on the right (`INS` for insert mode); the banner `FREE85 HOME` names the screen; the line `VERSION 2.10` beneath it names the firmware release and disappears as soon as you type; the underscore at the left of that line is the entry-line cursor; and the bottom row carries the soft-menu labels `MATH GRF VAR MEM SYS` for F1 through F5.

2nd ON turns the calculator off; the display goes completely blank. Press ON to switch it back on. Pressing ON while the calculator is already running does no

harm: a full-screen message answers ALREADY AWAKE, with the hint CLEAR OR EXIT, and either of those keys returns you to where you were.

1-2 The home screen and the entry line

Whatever you type appears on the entry line at the cursor. Press ENTER and the expression is evaluated: the result appears in the middle of the screen, introduced by =. Type 2 + 3 ENTER and the entry line shows 2+3 with = 5 below it.

The expression stays on the entry line after evaluation. That is deliberate: you can keep typing to extend it and press ENTER again, or press CLEAR to start fresh. Chapter 3 covers the expression language itself; this chapter sticks to the machinery around it.

1-3 Editing the entry line

The ◀ and ▶ keys move the cursor along the entry line, and DEL deletes the character just before the cursor. Type 1 2 3, press ◀ once, and press DEL: the 2 disappears, leaving 13.

The editor starts in insert mode, shown as INS in the status line, where typing pushes the characters after the cursor to the right. With 13 on the line, press ◀ and type 2: the line becomes 123.

2nd DEL toggles overwrite mode; the status indicator changes from INS to OVR. In overwrite mode typing replaces the character under the cursor, so the same ◀ 2 on 13 produces 12 instead. Press 2nd DEL again to return to insert mode.

CLEAR empties the entry line in one press.

1-4 Previous entries and the last answer

The ENTER key's shifted function is ENTRY, the previous-entry recall. Evaluate 2+3, press CLEAR, then press 2nd ENTER: the entry 2+3 reappears with the cursor at the end, ready to edit and re-evaluate.

The ▲ and ▼ keys walk the same history one step at a time, and the calculator keeps your four most recent entries. Evaluate 2+3 and then 5∗7, press CLEAR, and ▲ recalls 5∗7; ▲ again replaces it with 2+3, and each recall arrives with the cursor at the end. A repeated 2nd ENTER steps back exactly as ▲ does. ▼ returns towards the newest entry, and one step beyond it empties the line. A step

with nothing left to show answers the full-screen notice `NO MORE HISTORY`: ▲ past the oldest entry the calculator holds, ▼ beyond the emptied line, or either key on a fresh machine with no history yet. EXIT dismisses the notice and the recalled entry stays on the line.

The (-) key's shifted function is `ANS`, the most recent numeric result. With `5` as the last answer, press CLEAR, then 2nd (-) + 1 0 ENTER: the entry line reads `ANS+10` and the result is `= 15`. `ANS` can appear anywhere in an expression, as many times as you like, and always means the most recent numeric result.

Menus

1-5

The home screen's own soft-menu row is the top of the menu system. Its first page offers `MATH`, `GRF`, `VAR`, `MEM`, and `SYS` on F1 through F5; press MORE and the second page offers `LIST`, `MAT`, `VEC`, `STAT`, and `PGM`; press MORE again to cycle back to the first page.

Press F1 on the first page and the `MATH` menu takes over the screen:

The MATH menu opened with F1

A menu lists up to five items at a time, and the hint line `F1–F5 INSERT MORE` explains the keys: press the soft key matching an item to insert it into your entry (F1 here inserts `ABS(` and returns you to the home screen), or press MORE for the next page, which in the `MATH` menu starts with `SINH(` and `COSH(`.

Two keys back you out of any menu. EXIT goes up one level and leaves your entry untouched. 2nd EXIT is `QUIT`, which jumps straight back to the home screen from however deep you have wandered.

1-6 Mode settings

Press 2nd MORE to open the SYSTEM MODE screen:

The system mode screen

Three settings are listed, and the soft keys ANG FMT − + MEM adjust them:

- **Angle.** F1 (ANG) toggles between ANGLE RAD and ANGLE DEG. The choice is echoed in the home-screen status line and affects every trigonometric function.
- **Display format.** F2 (FMT) cycles FORMAT through AUTO, SCI, ENG, and FIX, then back to AUTO:
  - AUTO: ordinary decimal output, switching to an exponent for values outside the compact display range (the command catalog lists this setting under the names Normal and Float, which other calculators use for the same behaviour);
  - SCI: one digit before the decimal point and an explicit exponent, so 12345 displays as 1.2345E4;
  - ENG: one to three digits before the point and an exponent divisible by three, so 12345 displays as 12.345E3;
  - FIX: a fixed number of decimal places with half-up rounding, so at FIX 2 the result of 2/3 displays as 0.67.

  While FIX is selected, ▲ and ▼ change the number of decimal places, from FIX 0 up to FIX 11. The format changes how results are displayed, not the precision the calculator stores.

- **Contrast.** `CONTRAST 16` by default. F3 (–) lowers the setting one step at a time and F4 (+) raises it; the number updates as you press.

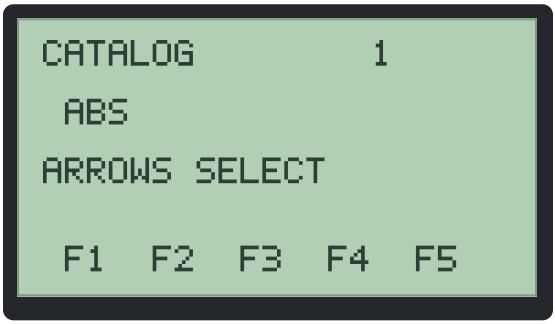
F5 (`MEM`) opens the memory browser from here; chapter 18 covers it. Press EXIT to leave the mode screen and return home.

Two settings that other calculators keep on a mode screen live on the graph screen instead: the graph type and the graph coordinate readout. Press 2nd MORE on the graph screen to open its format pages, then MORE until the `GRAPH MODE` page appears; its soft keys `FN POL PAR DEQ GC` choose between function, polar, parametric, and differential-equation graphing and toggle the coordinate readout. Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence) covers the format pages, and the graphing chapters that follow it cover each graph type. The complex and vector display choices live in their editors the same way: the complex editor of Chapter 11 (Complex Numbers) converts between forms with its `RECT` and `POLAR` soft keys, and the vector editor of Chapter 13 (Matrices and Vectors) shows its coordinate form as a tag, `RECTV` on a fresh machine.

The catalog and the custom menu

1-7

Every function you can call lives in one alphabetical list, the catalog. Press 2nd CUSTOM to open it:



The catalog, opened with 2nd CUSTOM

The header shows the page number and the current item (the list starts at `ABS`), with the hint `ARROWS SELECT`. ▲ and ▼ move through the list, and ENTER pastes the highlighted item into your entry line: 2nd CUSTOM ▼ ▼ ENTER

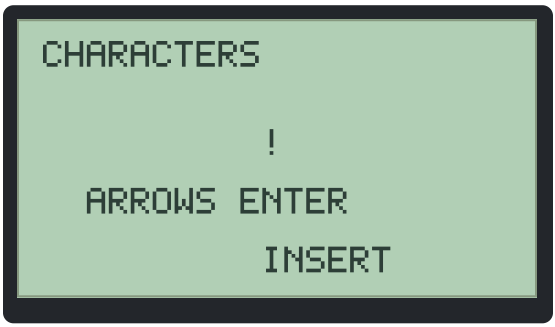
pastes the highlighted item into your entry line: [2nd] [CUSTOM] [▼] [▼] [ENTER] pastes `ACOSH(` at the home screen. [EXIT] leaves the catalog without choosing anything.

While an item is highlighted, pressing one of [F1] through [F5] assigns it to that slot of the custom menu, and the screen confirms with a message such as `ASSIGNED F2`.

Press [CUSTOM] (unshifted) to open the custom menu itself. Its five slots come preloaded with `ABS`, `EXP`, `LIS`, `SQR`, and `STA` as soft labels, and the screen explains itself: `F1–F5 RUN SLOT` and `MORE: CATALOG`. Pressing a soft key inserts that slot's function ([CUSTOM] [F1] pastes `ABS(`), and [MORE] jumps to the catalog so you can reassign slots. Keep your five most-used functions here and they are always two keypresses away.

1-8 The character palette

Press [2nd] [0] (`CHAR`) to open the character palette, titled `CHARACTERS`. It shows one character at a time, starting from the space character, so the middle of the screen looks empty until you move:



The character palette, one step right of the space character

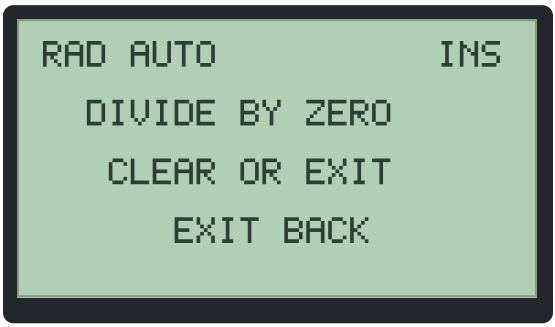
All four cursor keys step through the character set: [▶] and [▼] go forward, [◀] and [▲] go backward, and the ends wrap around (stepping back from the space character lands on the c cedilla at the far end). The hint `ARROWS ENTER` and the label `INSERT` say the rest: press [ENTER] to insert the shown character into your entry line, or [EXIT] to leave with the entry untouched. One step right of the space character is `!`, so [2nd] [0] [▶] [ENTER] types `!`.

The palette holds fifty-four characters. The space character and the twenty-five punctuation marks that follow it, from `!` to `_`, come first; after them the palette continues into twenty-four Greek letters (the capitals Alpha, Beta, Gamma, Delta, Theta, Lambda, Xi, Pi, Sigma, Phi, Psi, and Omega, then the same alphabet in lowercase with mu in place of Xi) and finishes with four international characters: e acute, n tilde, u umlaut, and c cedilla. The Greek and international characters insert exactly as the punctuation does: twenty-six steps right of the space character is the capital Alpha, and ENTER there appends it to your entry. Appendix A catalogues the additions as `Greek-characters` and `international-characters`. Chapter 9 covers the strings editor and its operations in full.

When something goes wrong

1-9

Errors are full-screen and polite. Type 1 ÷ 0 ENTER:



The divide-by-zero error screen

The screen keeps the status line and shows three lines: the error name `DIVIDE BY ZERO`, the hint `CLEAR OR EXIT` beneath it, and `EXIT BACK` at the bottom. Press CLEAR or EXIT and you are back on the home screen with `1/0` intact and the cursor at the end, so you can fix the mistake instead of retyping it. Appendix C lists every error message.

CHAPTER

2

Variables and Stored Data

A calculator you cannot save numbers in is only half a calculator. This chapter covers everywhere Free85 keeps a value for you: the twenty-six named variables `A` through `Z`, the five quick numeric memories `M1` through `M5`, the reserved names the system maintains itself, and the typed object store that holds all of it behind the scenes. Chapter 18: Memory Management continues the story with the memory browser, where stored objects are inspected and deleted.

Storing a value

2-1

The [STO▶] key types the store arrow, which appears on screen as `->`. An expression of the form `value->name` evaluates the value and stores it in the named variable. Variable names are single letters, typed with the [ALPHA] modifier followed by the key carrying that letter above it, so [ALPHA] [LOG] types `A`.

Type [5] [STO▶] [ALPHA] [LOG] [ENTER]:

Storing 5 in the variable A

The entry line reads `5->A`, and the result `= 5` confirms both the value and the store. The stored value also becomes `ANS`, exactly as if you had evaluated `5` on its

own.

The left-hand side can be any expression; it is evaluated first and the result is what gets stored. `2+3->C` stores `5` in `C`, and `PI->C` stores the full fourteen-digit value `3.1415926535898`. The arrow is an ordinary entry-line character, so you can cursor back and edit either side of it before pressing [ENTER].

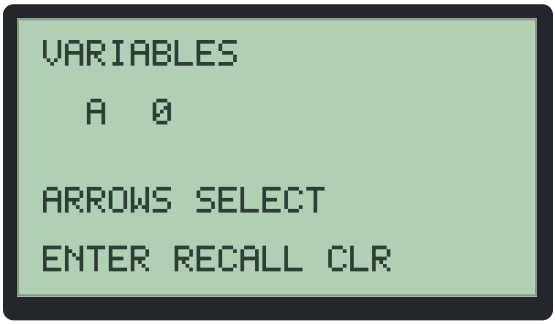
Names are exactly one letter. Storing to a two-letter name such as `AB` answers with the `SYNTAX ERROR` screen, and so does storing to a lowercase letter such as `a` (with one exception, `x`, described under reserved names below). Storing to a variable that already holds a value simply overwrites it; the previous value is gone.

2-2 Recalling a value

To use a stored value, type its name wherever a number could appear. With `5` in `A`, typing [ALPHA] [LOG] [×] [3] [ENTER] puts `A*3` on the entry line and answers `= 15`. A name can appear as often as you like in one expression, and every variable you have never stored to reads as `0`.

2-3 The variables browser

The [STO▶] key's shifted function is `RCL`. Press [2nd] [STO▶] and the `VARIABLES` browser takes over the screen:



The variables browser

The same screen opens from `VARS` ([2nd] [3]) and from the home screen's `VAR` soft key ([F3]).

The browser shows one variable at a time, starting at `A`, with the hint `ARROWS SELECT` beneath it. All four cursor keys step through the alphabet, [▶] and

▼ forward, ◀ and ▲ backward, wrapping at either end: one press of ◀ from A lands on Z. Press ENTER and the selected letter is pasted into your entry line back on the home screen, which is handy when you have forgotten which key holds which letter. EXIT leaves without pasting anything.

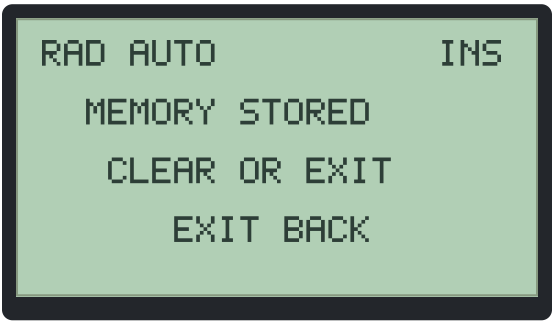
The readout beside the selected letter shows the variable's stored value, so with 5 in A the browser reads A 5. Pressing CLEAR performs the CLR action named on the bottom line: it zeroes the selected variable in place, and the readout updates to 0 immediately. Variables can also be deleted from the memory browser described in chapter 18.

The five numeric memories

2-4

The shifted functions of the soft keys are five one-press memories: M1 through M5 on 2nd F1 through 2nd F5. They work from the home screen and need no names.

- **With an expression on the entry line**, the memory key evaluates it and stores the result in that memory. Type 4 2 2nd F1:



The memory-stored notice

  The full-screen notice MEMORY STORED confirms, and CLEAR or EXIT returns you to the home screen with your entry intact. If the entry does not evaluate, the notice is MEMORY ERROR and nothing is stored.

- **With an empty entry line**, the memory key recalls: press CLEAR then 2nd F1 and the home screen shows = 42 with nothing on the entry line. The recalled value becomes ANS, so 2nd (-) carries it straight into your next expression.

The five memories are independent of each other and of `A` through `Z`. On a fresh machine each one recalls `0`. They survive switching the calculator off and warm restarts, and they are only emptied by a full reset from the memory browser (chapter 18).

2-5 Reserved names

A few names are special:

- **`A` through `Z`** always exist. They are the object store's reserved real-number entries, present from first boot with the value `0`, and they cannot be removed, only cleared. Deleting one from the memory browser resets its value to `0` and keeps its directory entry.
- **`x` and `X`** are the same variable, the graph variable. The [x-VAR] key types `X` in one press, and [ALPHA] [x-VAR] types the lowercase `x`; both spellings read and store the same value, so `5->x` followed by `X` answers `= 5`. The graphing chapters, beginning with chapter 4, use this variable as the plotting coordinate. Lowercase `x` is the only lowercase letter accepted in a variable name.
- **`ANS`** ([2nd] [(-)]) always names the most recent numeric result. It is maintained by the calculator and is not a storage target: `5->ANS` answers `SYNTAX ERROR`.

2-6 The typed object store

Underneath all of this, every stored item lives in one typed object store, kept in the store format introduced with Free85 2.0; the memory browser's banner names the running release, `MEMORY 2.10`. The store has room for named objects of eleven kinds (real numbers, complex numbers, lists, matrices, vectors, strings, equations, programs, user constants, graph databases, and pictures), its names can run to eight characters, and the browser names every entry's kind with a word: `A` is listed with `TYPE REAL`, and a stored picture with `TYPE PICTURE`. The one-letter rule above belongs to real-number variables; the longer names go with the other object types.

Beyond the reserved reals, the store fills through the calculator's own workflows rather than through the store arrow. A fresh machine carries the twenty-six reserved reals and nothing else, and the chapters ahead add to them: Chapter 8 (Physical and User Constants and Conversions) creates user constants, and the graphing chapters, beginning with chapter 4, store graph databases and pictures.

Lists, matrices, vectors, strings, complex working values, and programs never become named objects at all, though the store format keeps a type reserved for each. This is a deliberate design: each lives in its own home, such as the two working lists and result list of Chapter 12 (Lists) or the four program slots of Chapter 16 (Calculator Programming), which keep their contents when you leave but never appear in the memory browser. The stored graph equations likewise stay in their three slots `Y1`, `Y2`, and `Y3` outside the store. Chapter 18 tours the browser that lists every object with its type and exact size, along with the store's capacity and accounting rules.

Deleting stored data

2-7

There is no delete operation on the home screen; storing `0` over a variable is the quick way to neutralise it. Proper deletion, including per-object sizes and bulk clears, lives in the memory browser, which is the subject of chapter 18.

CHAPTER

3

Mathematics, Calculus, and Comparisons

This is the book's core reference for calculating with single numbers: the arithmetic operators and their precedence, powers and roots, logarithms, trigonometry, hyperbolic functions, factorials and combinatorics, the numeric utility functions, the comparison operators, and the numerical-calculus commands that analyse a stored function. Every example below was run on a fresh machine, and every result is quoted exactly as the calculator displays it. Free85 works in fourteen significant decimal digits throughout; the display format only changes how a result is presented, never what is stored (see the mode screen in Chapter 1: Operating the Calculator).

Finding the functions

3-1

The MATH menu (F1 from the home screen, or the MATH legend on 2nd ×) holds the most-used scalar functions:

The MATH menu opened with F1

Its first page carries ABS(, SQRT(, FACT(, NPR(, and NCR(, and its second page the hyperbolic family. Everything else in this chapter lives in the catalog (2nd CUSTOM), described in chapter 1. Wherever a function is filed, there is always a

shortcut: every name in this chapter can simply be typed letter by letter with [ALPHA].

3-2 Arithmetic and the order of operations

The four arithmetic keys [+] [-] [×] [÷] type the characters +, −, ∗, and / on the entry line, and [^] types ^ for powers. Typing [2] [+] [3] [ENTER] answers = 5; 2∗3 answers = 6 and 8/2 answers = 4.

Because the arithmetic is decimal throughout, results that look exact are exact: 0.1+0.2 answers = 0.3, and 12.5−2.75 answers = 9.75, with no binary floating-point noise in the last digit.

The [(-)] key is the unary minus. It types the same − character as the subtraction key, and the calculator reads the sign from context: −5 on its own answers = −5, 2∗−3 answers = −6, and 5−−3 (a subtraction followed by a negation) answers = 8. Because both keys insert the same character, you can use whichever is under your thumb; the expression means the same thing either way.

Expressions follow the usual order of operations:

- ^ binds tightest: 2∗3^2 answers = 18, not 36.
- Unary minus binds looser than ^: −3^2 answers = −9. When you mean the square of a negative number, parenthesise it: (−2)^2 answers = 4.
- ∗ and / come next, then + and −: 2+3∗4 answers = 14.
- Operators of equal rank evaluate left to right: 10−3−2 answers = 5.
- ^ chains right to left: 2^3^2 is 2^(3^2) and answers = 512.
- Parentheses override everything, exactly as written.

Multiplication can be implicit: 2PI answers = 6.2831853071796 (the π legend on [2nd] [^] types the constant PI), 2(3+4) answers = 14, and 3X multiplies by the graph variable. Implicit multiplication has the same precedence as the ∗ key, so 6/2PI evaluates left to right as (6/2)∗PI and answers = 9.4247779607694; parenthesise the denominator when you mean 6/(2PI).

For very large and very small numbers, the [EE] key types the exponent marker E: 1E−3 answers = 0.001 and 1E3+2 answers = 1002. (That is E between digits; standing alone, E names Euler's constant instead, as the logarithms section shows.) Exponents run from −128 through 127, and results that overflow that

range stop with the NUMERIC OVERFLOW error screen rather than silently losing precision.

Powers and roots

3-3

The [x²] key types ^2, so [3] [x²] [ENTER] puts 3^2 on the entry line and answers = 9. It is a plain piece of entry-line text; you can cursor back into it and edit it like anything else you typed.

The ^ operator takes any real exponent, by one of two routes, and which route it takes is worth knowing because it decides how exact the answer is.

An exponent that is a whole number in the signed 16-bit range is done by repeated squaring, which is exact: 2^9 answers = 512, 2^18 answers = 262144, 2^−1 answers = 0.5, 10^15 answers = 1E15, and the exponent may be any expression that evaluates to such a whole number, so 2^(3∗3) answers = 512. 0^0 answers = 1.

Any other real exponent, on a positive base, is done as EXP(y∗LN(x)). That is slower, because it runs a logarithm and an exponential, and it is approximate in the last digits: 2^0.5 answers = 1.4142135623734 where SQRT(2) answers = 1.4142135623731, and 27^(1/3) answers = 2.9999999999993 rather than a flat 3. Neither is wrong; they are different calculations of the same number, and the exact route is the one that goes through whole exponents.

Three cases are refused rather than approximated. A negative base with a fractional exponent has no real value, so (−2)^0.5 answers DOMAIN ERROR. Zero to a negative power is a division by zero, so 0^(−1) answers DIVIDE BY ZERO. A result too large for the numeric range answers NUMERIC OVERFLOW.

Four function keys cover the most common powers and roots:

- **Square root.** [2nd] [x²] (the √ legend) inserts SQRT(. SQRT(81) answers = 9, and SQRT(2) answers = 1.4142135623731. Negative arguments answer DOMAIN ERROR.
- **Powers of ten.** [2nd] [LOG] (the 10^x legend) inserts TEN(, which raises 10 to any real power. It works through logarithms, so its results are fourteen-digit approximations: TEN(3) answers = 999.99999999938. When the exponent is a whole number, typing 10^3 instead takes the exact route above and answers = 1000.

- **The exponential function.** 2nd LN (the e^x legend) inserts `EXP(`. `EXP(1)` answers = `2.7182818284583` and `EXP(2)` answers = `7.3890560989266`.
- **Reciprocal.** 2nd EE (the x^−1 legend) inserts `1/(`, so 2nd EE 4) ENTER evaluates `1/(4)` and answers = `0.25`.

For other roots, the catalog function `ROOT(x,n)` takes the nth root of x: `ROOT(27,3)` answers = `2.9999999999993`, again a fourteen-digit approximation computed through logarithms. `ROOT(` requires a positive x; `ROOT(−8,3)` answers `DOMAIN ERROR`, so take the root of the absolute value and reapply the sign yourself when you need the odd root of a negative number.

3-4 Logarithms

LN inserts `LN(`, the natural logarithm, and LOG inserts `LOG(`, the base-ten logarithm:

- `LN(2)` answers = `0.69314718056122`.
- `LOG(1000)` answers = `3` and `LOG(2)` answers = `0.30102999566454`.

Both functions require a positive argument; `LN(0)` answers `DOMAIN ERROR`. `LN(` and `EXP(` are inverses of one another, as are `LOG(` and `TEN(`, up to the fourteen-digit arithmetic: `LN(E)` answers = `1.0000000000006` because the constant `E` is itself stored to fourteen digits.

3-5 Trigonometry

The SIN, COS, and TAN keys insert `SIN(`, `COS(`, and `TAN(`. Their inverses sit on the same keys behind 2nd: 2nd SIN (the `SIN^−1` legend) inserts `ASIN(`, 2nd COS inserts `ACOS(`, and 2nd TAN inserts `ATAN(`.

All six obey the angle mode in the status line, set from the mode screen described in chapter 1. In `RAD` mode (the fresh-boot default):

- `SIN(PI/6)` answers = `0.5`.
- `TAN(PI/4)` answers = `1`.
- `ASIN(1)` answers = `1.5707963267949`, which is pi over two.

In `DEG` mode (2nd MORE F1, status line `DEG AUTO`):

- `SIN(30)` answers = `0.5`.
- `TAN(45)` answers = `1`.

- `ASIN(0.5)` answers `= 30` and `ATAN(1)` answers `= 45`.

The fourteen-digit arithmetic shows itself at the edges: in radians, `SIN(PI/2)` answers `= 0.9999999999939` rather than `1`, because `PI` itself is a fourteen-digit value; in degrees, `COS(60)` answers `= 0.49999999999989`. The tiny error is real, not a display artefact, and it is why results you know should be round sometimes come out a hair off in the last digits.

A large angle costs no more than a small one. The circular functions reduce their argument by a quotient rather than by repeated subtraction, so `SIN(400)` answers `= −0.85091935964129` and `SIN(999999)` answers `= −0.97735203155764` as readily as `SIN(1)` does. The supported range is one million radians, or one hundred million degrees, and across it `SIN` and `COS` are held to about 1E-7. Beyond the boundary a fourteen-digit input no longer fixes the angle's phase well enough to be worth reporting, and rather than return a plausible number the machine says so: `SIN(1.1E6)` answers `PRECISION LOST`.

Each trigonometric function takes exactly one argument; `SIN(1,2)` answers `SYNTAX ERROR`. Out-of-range inverse arguments, such as `ASIN(2)`, answer `DOMAIN ERROR`. To convert an angle between units explicitly rather than switching modes, the conversion functions `RAD(` and `DEG(` are available from the conversions menu: `RAD(180)` answers `= 3.1415926535898` and `DEG(PI)` answers `= 180`. Chapter 8: Physical and User Constants and Conversions covers that menu in full.

Hyperbolic functions

3-6

The `MATH` menu's second page carries `SINH(`, `COSH(`, `TANH(`, `ASINH(`, and `ACOSH(`. The sixth member, `ATANH(`, lives in the catalog. All six take their argument as a plain number, unaffected by the angle mode:

- `SINH(1)` answers `= 1.1752011936434`.
- `COSH(1)` answers `= 1.5430806348149`.
- `TANH(1)` answers `= 0.76159415595567`.
- `ASINH(1)` answers `= 0.88137358702064`.
- `ACOSH(2)` answers `= 1.316957896926`.
- `ATANH(0.5)` answers `= 0.54930614433465`.

Domain rules follow the mathematics: `ACOSH(` needs an argument of at least 1, so `ACOSH(0.5)` answers `DOMAIN ERROR`, and so does `ATANH(1)`.

3-7 Factorials, permutations, and combinations

`FACT(`, on the `MATH` menu's first page, computes the factorial (elsewhere `factorial` or `x!`):

- `FACT(0)` answers `= 1` and `FACT(5)` answers `= 120`.
- The argument must be a whole number from 0 through 69: `FACT(2.5)` and `FACT(−1)` answer `DOMAIN ERROR`.
- `FACT(69)` answers `= 1.7112245242814E98`, the largest factorial the numeric range holds; `FACT(70)` answers `NUMERIC OVERFLOW`.

`NPR(n,r)` counts permutations, ordered selections of `r` items from `n`, and `NCR(n,r)` counts combinations, where order does not matter:

- `NPR(5,2)` answers `= 20`.
- `NCR(5,2)` answers `= 10`.
- `NCR(20,10)` answers `= 184756`.

Both expect whole numbers with `r` no larger than `n`; `NCR(2,3)` answers `DOMAIN ERROR`.

3-8 Numeric utilities

A family of utility functions rounds out the scalar toolkit. Apart from `ABS(` on the `MATH` menu's first page, none of them sit on a menu, so type the names with **ALPHA**, paste them from the catalog, or keep your favourites on the custom menu (chapter 1):

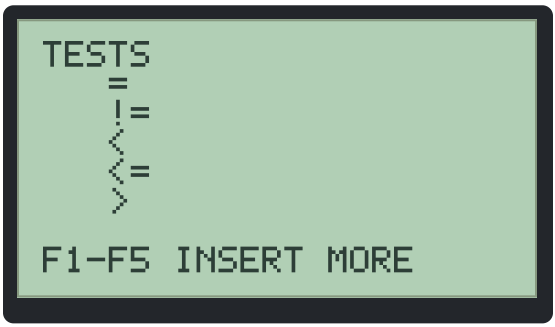
- **`ABS(`** strips the sign: `ABS(−7)` answers `= 7`.
- **`INT(`** keeps the whole-number part, truncating toward zero (elsewhere `iPart`): `INT(12.9)` answers `= 12` and `INT(−12.9)` answers `= −12`. Free85 has no separate floor-style `int` that would round `−12.9` down to `−13`.
- **`FRAC(`** keeps the fractional part, with the sign of its argument (elsewhere `fPart`): `FRAC(12.75)` answers `= 0.75` and `FRAC(−12.75)` answers `= −0.75`.
- **`ROUND(x,n)`** rounds to `n` decimal places, 0 through 11: `ROUND(PI,4)` answers `= 3.1416`.
- **`SIGN(`** answers `= 1`, `= 0`, or `= −1`: `SIGN(−3)` answers `= −1` and `SIGN(0)` answers `= 0`.

- **MOD(x,y)** is the remainder of x divided by y (elsewhere mod), with the sign taken from x: MOD(17,5) answers = 2, MOD(−7,3) answers = −1, and MOD(7,−3) answers = 1. MOD(5,0) answers DIVIDE BY ZERO.
- **GCD(and LCM(** work on whole numbers: GCD(84,30) answers = 6 and LCM(6,8) answers = 24.
- **MIN(and MAX(** take exactly two arguments: MIN(−2,3) answers = −2 and MAX(−2,3) answers = 3. (MIN(1,2,3) answers SYNTAX ERROR; nest calls for longer lists.)
- **PCT(x,p)** answers p percent of x (elsewhere percent): PCT(200,15) answers = 30 and PCT(80,25) answers = 20.
- **ROOT(x,n)**, described under powers and roots above, is the nth-root companion to this family.
- **RAND()** returns a pseudo-random value between 0 and 1 with four decimal places. The generator is deterministic: a fresh machine always answers = 0.7968 first and = 0.8984 second, and the sequence carries on from wherever it left off. RANDI(low,high) returns a whole number from low through high inclusive, drawn from the same sequence: RANDI(4,9) on a fresh machine answers = 6. Treat both as repeatable test sequences, not as a source of secrets.

Comparisons

3-9

Press 2nd 2 (the TEST legend) and the TESTS menu lists the six comparison operators:



The TESTS menu opened with 2nd 2

Page one offers =, !=, <, <=, and > on F1 through F5; press MORE for >=. Pressing a soft key inserts the operator into your entry and returns you to the home screen.

A comparison evaluates to a calculator boolean: `1` for true, `0` for false.

- `5=5` answers = `1` and `2=3` answers = `0`.
- `2!=3` answers = `1`.
- `2<3` answers = `1`, `2<=2` answers = `1`, `4>3` answers = `1`, and `2>=3` answers = `0`.

Note the spellings. Equality is the single = character; a doubled ==, which some languages and calculators use, answers `SYNTAX ERROR`, and so does the <> spelling of not-equal. The not-equal operator is `!=`, exactly as the `TESTS` menu inserts it.

Because the result is an ordinary number, comparisons combine freely with arithmetic: `(2<3)+(5>=5)` answers = `2`. Comparisons do not chain, though: `2<3<1` answers `SYNTAX ERROR`, so write the two comparisons separately. Chapter 16: Calculator Programming puts these operators to work in `If` and `While` conditions.

3-10 Numerical calculus

Free85's calculus commands analyse the *active graph equation* rather than taking an expression as an argument. This is a deliberate design: the short forms keep expressions comfortably inside the entry line. Store a function once, then evaluate, differentiate, integrate, and search it from the home screen, from programs, or from the catalog.

To store the function, type it on the home entry line using x-VAR for the variable and press GRAPH: x-VAR x² GRAPH stores `X^2` as `Y1` and plots it. Storing is all the calculus commands need, and GRAPH stores the entry line before the first column is drawn, so the plot need not finish: interrupt it with EXIT at any point and `EVAL(3)` still answers = `9`. Only an empty active slot leaves the commands

nothing to read, and then they answer `SYNTAX ERROR`. Press **EXIT** to return home; your entry is still on the line, so press **CLEAR** and put the commands to work:

EVAL(3) evaluating the stored X^2

With `X^2` stored as the active equation:

- **`EVAL(x)`** evaluates the active equation: `EVAL(3)` answers `= 9`.
- **`NDER(x)`** takes a central numerical derivative: `NDER(3)` answers `= 6`.
- **`FNINT(a,b)`** integrates over `[a,b]` by comparing composite Simpson estimates rather than trusting one: 32 panels, then 64, and 128 if those two do not yet agree. `FNINT(0,2)` answers `= 2.6666666666667`, the fourteen-digit 8/3. If the estimates will not settle inside that work budget the command answers `NO CONVERGENCE`, and an endpoint where the integrand is undefined answers `DIVIDE BY ZERO`. Neither is a failure of nerve: a refined-looking number that is quietly wrong is worse than a refusal you can act on.
- **`FMIN(a,b)` and `FMAX(a,b)`** search `[a,b]` and return the *location* of the extremum, not its value: `FMIN(-2,2)` answers `= 0.00011982342365967`, the numerical minimum of the parabola near zero, and `FMAX(-2,2)` answers `= -1.9997326856357`, closing in on the `-2` endpoint. Follow up with `EVAL(` on the answer when you want the value there; the small residuals are the honest output of the search.
- **`INTER(a,b)`** interpolates linearly between the endpoint values and returns the midpoint of that chord: with `X^2` stored, `INTER(0,2)` answers `= 2`, halfway between the endpoint values `0` and `4`.
- **`ARC(a,b)`** sums a 64-segment polyline approximation to the arc length: `ARC(0,1)` answers `= 1.4789246603137`.

Called with arguments alone, each of them reads the active equation, so a slot holding `EVAL(2)` in that form would have to evaluate itself. Give the command a slot number first and it reads that slot instead: `EVAL(slot,x)` and `NDER(slot,x)`, and likewise `FNINT(slot,a,b)`, `FMIN(slot,a,b)`, `FMAX(slot,a,b)`, `ARC(slot,a,b)` and `INTER(slot,a,b)`, with slots numbered from one. A slot holding `NDER(1,X)` therefore plots the derivative of slot 1 as a function of X, and one holding `FNINT(1,0,X)` plots its accumulated area, as Chapter 4: Cartesian Graphing, Drawing, Formats, and Persistence shows. One nested graph evaluation is available, so a slot may read another slot; a slot that reaches itself, directly or round a cycle, answers `RECURSION ERROR` while the unrelated slots carry on.

If you are arriving from another calculator's manual, the names map like this (Free85 spelling first, then the name elsewhere):

- `EVAL(` covers the expression-argument evaluators `eval` and `evalF`.
- `NDER(` covers the numerical derivatives `nDer` and `der1`. There is no separate second-derivative command, so for `der2` workflows apply `NDER(` to a stored derivative expression.
- `EVAL(` on a stored polynomial covers the polynomial evaluator `peval`; the polynomial's roots come from the `POLY` solver in Chapter 14: Equation, Polynomial, and Simultaneous Solving.
- `INTER(` and `ARC(` correspond to `inter` and `arc`.

The same analysis (roots, extrema, derivative, and integral, plus intersection between equations) is available graphically from the graph screen's soft keys, working on the plotted window instead of an interval you type. Chapter 4: Cartesian Graphing, Drawing, Formats, and Persistence covers that workflow, the `Y2` and `Y3` slots, and everything else about the graph screen.

Finally, **2nd** **CLEAR** (the `TOLER` legend) cycles the numeric tolerance through `1E−6`, `1E−8`, and `1E−10`, confirming each press with a `TOLERANCE CHANGED` notice; a fresh machine starts at `1E−6`. The root-hunting analyses of chapters 4 and 14 test their residuals against this setting.

3-11 Angle and fraction display

Angles in Free85 are plain decimal numbers, read as radians or degrees according to the `RAD/DEG` mode set on the mode screen of chapter 1 (elsewhere `Radian` and `Degree`). There is no degrees-minutes-seconds entry or display (elsewhere `→DMS`).

This is a deliberate design: an angle is always one plain number, so 30 degrees 15 minutes is typed as its decimal equivalent and results read the same way. The designed equivalents are ordinary functions on the conversions menu of chapter 8. `RAD(` and `DEG(`, shown in the trigonometry section above, convert a whole angle between the units, and the minutes-seconds pair `MINS(` and `SMIN(` steps a figure across the same sixty-to-one ratio that links arcminutes to degrees: `MINS(2)` answers `= 120`, `SMIN(90)` answers `= 1.5`, and `30+SMIN(15)` answers `= 30.25`, the decimal form of 30 degrees 15 minutes. Appendix A catalogues the entry side of this boundary as `DMS-entry` alongside the display command.

Fraction display sits on the same boundary: results never present as fractions (elsewhere `->Frac`), and `FRAC(` is the fractional-part utility of this chapter rather than a fraction display (`FRAC(12.75)` answers `= 0.75`).

CHAPTER

4

Cartesian Graphing, Drawing, Formats, and Persistence

This chapter is the full tour of the graph screen: storing equations in the three function slots, watching a plot draw, changing the window with the zoom keys, tracing along a curve, running the root and calculus analyses directly from the plot, and reading the table of values. It closes with drawing on a completed plot and with storing pictures and graph databases. Every key sequence and every quoted number below was run in the emulator on a fresh machine.

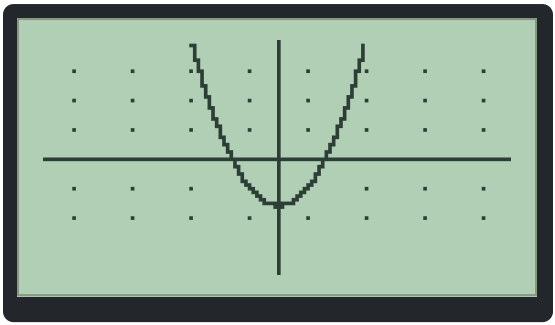
Free85 graphs in four modes; this chapter describes the default: real functions of the graph variable `X` (elsewhere `Func`). The polar, parametric, and differential-equation modes have their own chapters (5, 6, and 7) and share everything described here.

> **HARDWARE**
> all graphing in this chapter, including plot speed and the LCD captures, is validated in the emulator; physical hardware validation is reported separately.

Storing an equation and plotting it 4-1

There is no separate equation-entry screen: the home entry line is the equation editor. Type an expression in `X` using **x-VAR** (the graph variable is described in Chapter 2: Variables and Stored Data), then press **GRAPH**. The calculator saves

the entry line into the active function slot and starts plotting. Type x-VAR x² - 4 GRAPH and the parabola `X^2-4` appears:



The plot of X^2-4 in the standard window

The plot area is the whole 128 by 64 pixel display. The axes cross at the origin, and the faint dots are the grid, drawn at a fixed screen spacing rather than at unit intervals. The ordinary graph screen uses every key directly; the format and zoom panels add labelled soft-menu rows when opened.

A plot draws column by column from left to right, 128 samples in all. A single equation takes a few seconds and plots noticeably faster than three; a heavy expression slows every column. You can leave at any time: EXIT or CLEAR cancels the redraw and returns to the home screen with the equation loaded on the entry line. Cancelling costs only the picture: GRAPH stored the equation before the first column was drawn, so the home-screen calculus commands of Chapter 3: Mathematics, Calculus, and Comparisons, `EVAL(` among them, read it whether or not the plot was left to finish.

The plotter is deliberately hard to crash. Discontinuities, domain errors, and values outside the window leave gaps instead of stopping the plot: `1/X` plots both branches with a one-column gap at `X=0`, `SQRT(X)` plots nothing left of the origin, and adjacent samples are joined into a connected curve only when they are fewer than seventeen pixels apart, so a vertical asymptote is not smeared into a false vertical line. A graph-calculus command that reaches its own slot is refused rather than followed round: `NDER(1,X)` stored in slot 1, or `EVAL(2,X)` in slot 1 against `EVAL(1,X)` in slot 2, answers `RECURSION ERROR`, and that slot draws no curve while the slots beside it plot exactly as they would alone. Pointed at a different slot the same command is ordinary work, because one nested graph evaluation is

available: `NDER(1,X)` stored in slot 2 plots the derivative of slot 1 across the window.

Pressing **GRAPH** again on the graph screen replots. Pressing the `GRF` soft key (**F2** on the home screen's first menu page) opens the same graph screen.

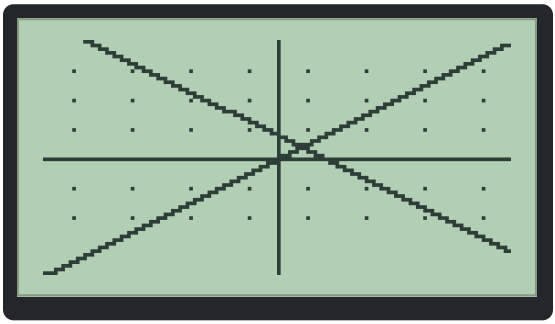
One side effect worth knowing: plotting evaluates the equation at every sample, and each evaluation stores the sample into the graph variable, so after a completed plot `X` holds the last sampled value (`X` on the entry line after the plot above answers `= 9.9999999999959`).

The three function slots

4-2

Free85 has three function slots, `Y1`, `Y2`, and `Y3`. **GRAPH** always saves the entry line into the *active* slot, and `Y1` is active on a fresh boot. To switch slots, press **2nd** **1**, **2nd** **2**, or **2nd** **3** *on the graph screen*: the calculator makes that slot active and returns to the home screen with the slot's stored text loaded on the entry line, ready to edit.

To plot two lines together: type **x-VAR** **GRAPH** to store `X` as `Y1`, press **2nd** **2** to switch to `Y2` (the entry line comes back empty), type **2** **-** **x-VAR**, and press **GRAPH**:



Y1=X and Y2=2-X plotted together

Both slots are now enabled and every replot draws both. A slot is enabled whenever it holds text: storing an empty entry line disables and clears the slot, so to switch `Y2` off, press **2nd** **2** on the graph screen, press **CLEAR**, then press **GRAPH**. Equation selection (elsewhere `FnOn` and `FnOff`) can also be changed without erasing text: open graph format with **2nd** **MORE**, press **MORE** for its

second page, then use F2, F3, or F4 to toggle Y1, Y2, or Y3. An enabled stored equation is on; a disabled stored equation remains available for later use.

4-3 The window and the zoom keys

The window is described by four values, the minimum and maximum of each axis: the default window runs from -10 to 10 on both axes, and each plotted column steps one 127th of the width. The window is changed with the zoom keys on the graph screen, each of which replots immediately:

- + zooms in (elsewhere ZIn): every window bound is halved, closing in on the origin by a factor of two. One press on the default window gives -5 to 5 on both axes.
- - zooms out (elsewhere ZOut): every bound is doubled, giving -20 to 20 from the default window.
- 2nd + restores the standard window (elsewhere ZStd): -10 to 10 on both axes, from wherever your zooming has taken you.
- 2nd - sets the square window (elsewhere ZSqr): -10 to 10 horizontally and -5 to 5 vertically. Because the LCD is 128 pixels wide and 64 tall, this window makes one unit the same length on both axes, so circles look like circles.

For the complete zoom panel, press 2nd GRAPH. Its three pages are:

- page 1: F1 ZBox, F2 ZIn, F3 ZOut, F4 ZStd, and F5 ZSqr;
- page 2: F1 ZDecm, F2 ZFit, F3 ZInt, F4 ZPrev, and F5 ZTrig;
- page 3: F1 stores the current window, F2 performs ZRcl, F3 or F4 selects factor 2 or factor 4 for later zoom-in and zoom-out, and F5 WIN opens the window editor described below.

Press MORE to cycle pages and EXIT to return to the plot. ZBox starts a movable cursor: position one corner with the arrow keys and press ENTER, then position the opposite corner and press ENTER again. This box is the free-form user-defined-zoom. The factor 2/factor 4 choices are the zoom-factors; they remain selected until changed. ZFit retains the horizontal range and derives a padded vertical range from the active curve. ZPrev exchanges the current and previous windows, while store and ZRcl provide a separate remembered window.

The current bounds are readable on the home screen as XMIN, XMAX, YMIN, and YMAX. They are read-only there; to type a window directly, use the editor below.

The window editor

4-4

Zoom keys reach a window by halving and doubling, which is quick and rarely exact. When the bounds matter, type them. From the graph screen press **2nd** **GRAPH** for the zoom panel, **MORE** twice for page 3, and **F5** (`WIN`):

The graph window editor with all four bounds

The four bounds are listed with a cursor on the selected one, and the soft keys are `XMN XMX YMN YMX SAVE`. **F1** to **F4** select a bound; typing puts a value on the entry line, and **ENTER** commits it to a private draft. The line above the soft keys prompts `TYPE VALUE; ENTER` while you work.

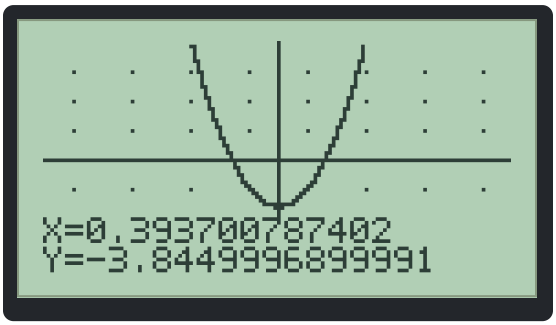
Two things make this safe to use on a window you care about. Drafts are private until you save: editing `XMIN` and then leaving changes nothing. And **F5** (`SAVE`) commits all four at once, after checking that `XMIN` is below `XMAX` and `YMIN` below `YMAX`. If they are not, the panel stays open with `DOMAIN ERROR` and the live window is untouched, so a half-entered window can never reach the plotter.

Values are expressions, not just digits, which is the point of typing them. `2*PI` is a legal `XMAX`, and so is `LN(100)`; the whole entry line is evaluated when you press **ENTER**. **EXIT** cancels an expression in progress, and pressing it again returns to the zoom panel without saving.

Trace

4-5

The ◀ and ▶ keys trace along the active equation (elsewhere `Trace`; there is no separate trace mode to switch on). The trace position starts at the centre column of the plot, each press moves it one column, and the readout at the bottom of the screen shows the exact coordinates. With `X^2-4` plotted, press ▶ twice:



Tracing X^2-4 two columns right of centre

The readout gives `X=0.393700787402` and `Y=-3.8449996899991`: the trace X values are the exact sample positions, spaced one 127th of the window width apart, and Y is the active equation evaluated there in full fourteen-digit precision. The trace stops at the left and right edges of the window. Two quirks: the readout is drawn over the bottom rows of the plot (press **GRAPH** to redraw cleanly), and there is no marker on the curve itself yet, so the readout is the trace.

Press ▲ or ▼ from an ordinary completed plot to start the free cursor at the centre of the screen. All four arrow keys move it independently of the curve, and its footer reports the corresponding window coordinate. **EXIT**, **CLEAR**, or **GRAPH** leaves free-cursor mode and redraws. Turning coordinate display off in graph format suppresses both trace and free-cursor footers.

Tracing also sets the reference position used by the analyses in the next section: the derivative is taken, and the root search begins, at the last traced `X` (which is `X=0` until you move the trace).

4-6 Analysis from the graph screen

The five function keys run the numerical analyses directly on the plotted equations, using the current window as the interval. Each publishes its result on the home screen. With `X^2-4` plotted:

- F1 **finds a root** of the active equation. The search starts from the traced position, then scans the window from the left edge for a sign change and closes in by bisection, so with `X^2−4` it answers `= −2`, the leftmost root in the window. A second line reports the residual, here `R=0`: the value of the equation at the reported root, which the tolerance setting (chapter 3) requires to be small.

The root of X^2-4 published on the home screen with its residual line

- F2 **finds a minimum and** F3 **a maximum**, searching the whole window and answering the *location* of the extremum: F2 with `X^2` plotted answers `= 0.00059911711827895`, the numerical minimum near zero, in the same honest-residual style as `FMIN(` and `FMAX(` in chapter 3.
- F4 **takes the derivative** at the traced position by central difference: with `2*X+3` plotted it answers `= 2`, and with `X^2` plotted, tracing three columns right of centre and pressing F4 answers `= 1.102362205`, twice the traced `X`.
- F5 **integrates** the active equation across the window by the same compared-estimate Simpson rule as `FNINT(`: with `X^2` in the standard window it answers `= 666.66666666667`, the fourteen-digit 2000/3, and it refuses in the same terms when the estimates will not settle.
- 2nd F1 **finds an intersection** of `Y1` and `Y2` (both slots must be enabled). With `Y1=X` and `Y2=2−X` it answers `= 1`, again with a residual line.

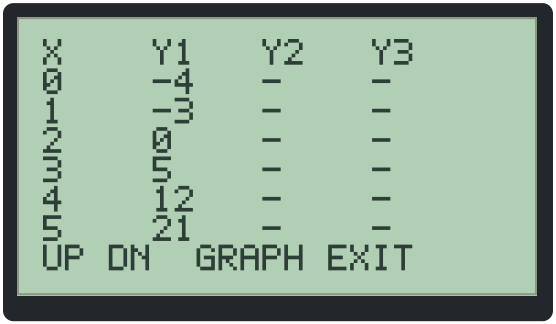
When a search fails, for instance F1 on `X^2+1`, which never crosses zero, the answer is the `NO NUMERIC RESULT` notice rather than a made-up number; CLEAR or EXIT dismisses it.

These are the graph-side counterparts of the home-screen calculus commands (`EVAL(`, `NDER(`, `FNINT(`, `FMIN(`, `FMAX()` described in chapter 3: same algorithms,

with the window standing in for the interval arguments. The GRAPH key's shifted function SOLVER runs the same hunt from its own workspace: with an expression on the home entry line, 2nd GRAPH stores it there, and the workspace's SOLV key searches between bounds you set. Chapter 14: Equation, Polynomial, and Simultaneous Solving covers the solving tools in full.

4-7 The table

Press MORE on the graph screen to open the table of values:



The table of values for Y1=X^2-4

The table shows six rows, an X column on the left, and a column per function slot. It starts at X=0 and steps by 1, so with X^2−4 stored the X column reads 0 through 5 and the Y1 column reads −4, −3, 0, 5, 12, 21. A disabled slot's column shows −, and a value that does not exist shows UNDEF: plot 1/X and open the table, and the X=0 row shows UNDEF with the reciprocals below it. Cells are five characters wide, so long values are truncated to fit (0.333 for a third).

The table keys are listed on its bottom line, UP DN GRAPH EXIT:

- ▲ and ▼ scroll by five rows: one press of ▼ restarts the table at X=5.
- + doubles the step and - halves it, so from the default the rows step 2, 4, 8 apart, or 0.5, 0.25 going the other way.
- GRAPH or EXIT leaves the table and replots the graph.

The start and step you reach this way survive leaving the table, so a scrolled table reopens where you left it, but neither value can be typed in directly: the four keys above are the only controls today.

Graph formats

4-8

Press 2nd MORE on the graph screen to open the persistent format panel. On its first page, F1 toggles AxesOn/AxesOff, F2 toggles CoordOn/CoordOff, F3 toggles LabelOn/LabelOff, F4 toggles GridOn/GridOff, and F5 selects DrawLine or DrawDot. Line mode joins adjacent valid samples; dot mode plots only the samples. . remains a quick grid toggle from the ordinary graph screen.

Press MORE for the second format page. F1 selects SimulG or SeqG: simultaneous mode samples every enabled equation at each X column, whereas sequential mode completes one equation before starting the next. Both produce the same final framebuffer. F2, F3, and F4 toggle Y1, Y2, and Y3; F5 moves directly to the zoom panel. A further press of MORE reaches the graph mode page of chapters 5 to 7, and the pages then cycle. EXIT applies the persistent settings and redraws.

Drawing on a graph

4-9

Press CUSTOM on a completed graph to open DRAW. The first page maps F1 through F5 to Line, Vert, Circ, TanLn, and Shade. Press MORE for PtOn, PtOff, PtChg, DrawF, and DrInv; press it again for the freehand–pen and ClDrw plus the picture controls.

Cursor tools begin in the centre. Move with the arrow keys. Line and Circ use ENTER once to fix their first point and again to draw; vertical, tangent, and point tools use one ENTER. The pen draws as you move. EXIT or CLEAR leaves a cursor tool. Shade and function drawing proceed incrementally and can be cancelled. ClDrw discards marks by redrawing the equations and axes.

Storing and recalling graphs

4-10

On DRAW page 3, F3 StPic stores the exact LCD image as PIC1, F4 RcPic recalls it, and F5 StGDB stores the equations, window, table, format, and mode settings as GDB1. Press MORE once more for page 4, RCG OVR OFF: F1 RcGDB restores that database and redraws, F2 OVR arms a picture overlay, and F3 OFF disarms it.

RcPic puts a stored picture on the screen, but the next plot clears it, which is no use for comparing two curves. OVR is the answer to that. It recalls PIC1 and holds

its 1,024 pixels as a one-shot underlay: the next graph draws *on top of* them instead of over a blank screen, so two plots share one picture. The underlay is spent once used, and the redraw after it clears normally.

The sequence for comparing two functions is therefore: plot the first, **CUSTOM** and **MORE** twice and **F3** to store it as `PIC1`, change the equation, then **CUSTOM** and **MORE** three times and **F2** (`OVR`) before pressing **GRAPH**. Keep the window identical between the two or the comparison is meaningless, which is one of the things the window editor above is for.

`PIC1` itself is never modified by any of this: the overlay reads it. **F3** (`OFF`) cancels an armed underlay if you change your mind, and arming `OVR` with no `PIC1` stored displays `NO PICTURE` and leaves nothing armed. Re-storing replaces the same named object, so repeated saves do not consume another directory entry.

Pictures and graph databases are native Free85 objects shown by the memory browser. They are not TI file-format or binary-compatible objects. Programs can call the same operations with hexadecimal `DRAW` codes; the complete code table is in `docs/Free85-graph-drawing-persistence.md`.

CHAPTER

5

Polar Graphing

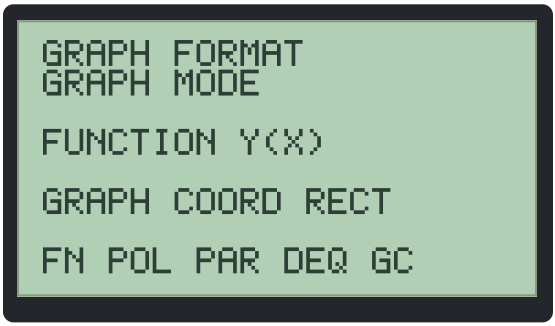
Polar graphing plots curves given as a radius in terms of an angle, the natural form for circles about the origin, spirals, and rose curves. The mode (elsewhere `Pol`*) sits on the same graph engine as Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence): the same window and zoom keys, the same format toggles, the* `.` *grid toggle, the same DRAW tools on* ***CUSTOM****, and the same* ***MORE*** *table. This chapter covers what the mode adds and changes, and every figure in it is quoted from the machine.*

> **HARDWARE**
> polar graphing, its plots, and the LCD captures are validated in the emulator; physical hardware validation is reported separately.

Switching the graph mode

5-1

The graph mode lives on the third page of the graph format panel. On the graph screen, press **2nd** **MORE** to open graph format, then press **MORE** twice:



The graph mode page with the function mode selected

The panel keeps its GRAPH FORMAT banner and titles this page GRAPH MODE. The middle line names the current mode, FUNCTION Y(X) on a fresh machine, and the line below it shows the coordinate readout setting, GRAPH COORD RECT. The soft keys are FN POL PAR DEQ GC: F1 through F4 select the function, polar, parametric, and differential-equation modes, and any of the four closes the panel and replots immediately in the new mode. F5 is the coordinate toggle (elsewhere PolarGC): it switches the line between GRAPH COORD RECT and GRAPH COORD POLAR and stays on the page, and EXIT then returns to the plot. What the toggle changes is described under tracing below.

5-2 Entering a polar equation

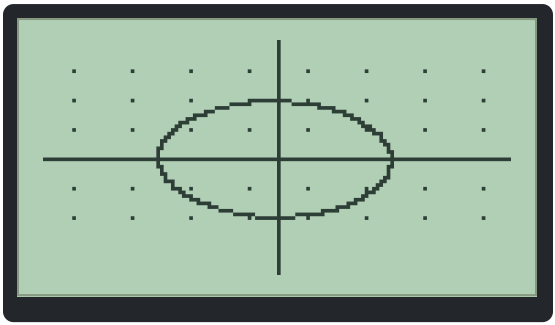
Press F2 on the mode page for polar. As in chapter 4 there is no separate equation screen: the home entry line is the editor, GRAPH stores the line into the active slot, and 2nd 1, 2nd 2, and 2nd 3 on the graph screen switch slots. The difference is the meaning of slot 1: it holds the radius as a function of the angle.

The angle is typed with x-VAR. The equation language has no theta variable: x-VAR inserts the character X, the equation is stored exactly as typed, and in polar mode the plotter reads that X as the sweep angle. The character palette of Chapter 1 (Operating the Calculator) does carry both theta glyphs, but a palette theta is only a character; it does not name the angle. A rose curve is therefore stored as 5*SIN(3X).

5-3 The sweep and the window

Polar mode has no angle-range settings. The plotter always sweeps the angle through one full revolution in 128 samples: 0 to 2 pi in RAD mode, and 0 to 360 in DEG mode, so the angle mode on the system mode screen (chapter 1) changes where the samples fall. The window and zoom keys of chapter 4 control the viewport only; zooming never stretches or trims the sweep.

To draw the worked example, select polar on the mode page, press **EXIT**, type **5**, and press **GRAPH**:



The polar plot of r=5 in the standard window

A constant radius sweeps a circle of radius 5. In the standard window it draws as an ellipse because the pixels are not square; **2nd** **-** sets the square window (`ZSqr`, chapter 4) and rounds it. Replacing slot 1 with `5*SIN(3X)` plots a three-petal rose. Plotting is the same cancellable column-by-column redraw as chapter 4, one sample per column: **EXIT** or **CLEAR** cancels mid-plot and returns to the home screen with the equation on the entry line.

Tracing a polar curve

5-4

◀ and ▶ trace along the sweep, starting from its centre sample and moving one sample per press. The readout follows the curve around the origin rather than left to right. Its labels are always `X=` and `Y=`; what fills them is the coordinate setting from the mode page:

- With `GRAPH COORD RECT`, the readout is the Cartesian point. On the circle above, one press of ▶ shows `X=-4.9862524284893` and `Y=-0.37071118578787`.
- With `GRAPH COORD POLAR`, the same position shows the radius in the `X=` line and the angle in the `Y=` line: `X=5` and `Y=3.2158035036746` in `RAD` mode, or `Y=184.25196850393` for the same press in `DEG` mode. The labels do not change with the setting, so remember which reading is selected.

Each trace step recomputes its sample, which takes a noticeable moment, and keypresses that arrive while it works are dropped, so trace at the calculator's pace rather than by holding the key. The free cursor (▲ or ▼, chapter 4) is

unchanged in polar mode: it reports plain window coordinates whatever the coordinate setting says.

5-5 The polar table

MORE on the graph screen opens the table of chapter 4 with the same six rows, the same headings `X Y1 Y2 Y3`, and the same scrolling and step keys. In polar mode the `X` column holds the angle and `Y1` the radius evaluated there, starting at 0 and stepping by 1. With the circle stored, the `Y1` column reads `5` in every row; with the rose `5*SIN(3X)` in `RAD` mode it reads `0`, `0.705`, `−1.39`, and so on down the rows, truncated to the five-character cells of chapter 4.

5-6 Analysis in polar mode

The five function keys and 2nd F1 remain live, but they treat the stored radius as an ordinary function of the angle over the *window* interval `XMIN` to `XMAX`, not over one revolution. With the circle `5` stored and the standard window:

- F4 answers `= 0`, the derivative of a constant radius;
- F5 answers `= 100`, the integral of 5 across the 20-unit window;
- F1 answers the `NO NUMERIC RESULT` notice, since the radius never crosses zero;
- `EVAL(2)` on the home screen answers `= 5`, the radius at angle 2.

None of these is an arc length or a swept area; read them as chapter 3 calculus applied to the radius function. The intersection search 2nd F1 needs slot 2 enabled, and since slot 2 never plots in polar mode the search has no real use here.

5-7 What the mode remembers

Each graph mode keeps its own equations, enabled slots, active slot, window, table position, and coordinate setting. When you switch modes, the outgoing mode's state is written to a named object in the store, `GPOL` for polar (the others are `GFUNC`, `GPAR`, and `GDEQ`), and switching back restores it exactly. The memory browser of Chapter 18 (Memory Management) lists it as `TYPE GRAPH DB` with `SIZE 213`, and DEL on it resets that one mode to factory state the next time the mode is entered. Note that the object is written when you *leave* the mode, so the mode you are currently in has no entry to delete yet.

Boundaries worth knowing

5-8

Slots 2 and 3 never plot in polar mode, but they are not inert: a slot holding text stays enabled, and the table then shows `UNDEF` in its column for every row. Clear a leftover slot the chapter 4 way, by selecting it with **2nd** **2** or **2nd** **3** on the graph screen and pressing **GRAPH** with the entry line empty, remembering that this also erases the slot's text. Appendix A catalogues this chapter's workflow as `polar-editor`, `polar-plot`, `polar-trace`, `polar-table`, and `polar-analysis`.

CHAPTER

6

Parametric Graphing

Parametric graphing plots curves whose horizontal and vertical coordinates are each given as a function of a shared parameter, the form that handles projectile paths, Lissajous figures, and any curve that doubles back on itself. The mode (elsewhere `Param`) runs on the shared graph engine of Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence), and every number quoted in this chapter comes straight from the machine.

> **HARDWARE**
> parametric graphing, its plots, and the LCD captures are validated in the emulator; physical hardware validation is reported separately.

Switching to parametric mode 6-1

Open the graph mode page exactly as in Chapter 5 (Polar Graphing): press **2nd** **MORE** on the graph screen, then **MORE** twice to reach `GRAPH MODE`, and press **F3** (`PAR`). The middle line changes to `PARAM X(T),Y(T)`, the panel closes, and the screen replots in the new mode.

Entering a coordinate pair 6-2

The three slots take on fixed roles: slot 1 is the horizontal coordinate x(t), slot 2 is the vertical coordinate y(t), and slot 3 is never plotted. The two equations form a single pair; there is no second pair. As everywhere else, the home entry line is the editor, **GRAPH** stores it into the active slot, and **2nd** **1** and **2nd** **2** on the graph screen switch slots.

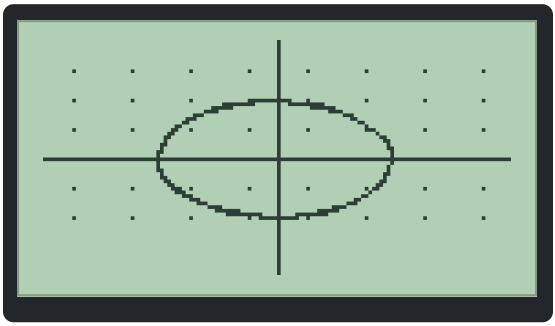
The parameter is typed with **x-VAR**, which inserts the character `X`; the equations are stored exactly as typed, and in this mode the plotter reads that `X` as t. Both

slots must hold text before anything draws: with only x(t) stored, GRAPH completes an apparently normal plot that draws no curve at all, and no notice points at the empty slot, so an axes-only result in this mode usually means y(t) is missing.

6-3 The parameter range and the window

There are no parameter-range settings. The parameter always runs from `XMIN` to `XMAX` in 128 samples: the window's horizontal bounds double as the sweep, so the zoom keys of chapter 4 change the viewport and the parameter range together. In the standard window t runs from -10 to 10.

A first pair worth trying is `X` in slot 1 and `X^2` in slot 2, which traces the parabola of chapter 4 point for point. The worked example is a circle: store `5*COS(X)` as slot 1, switch with 2nd 2, store `5*SIN(X)`, and let the plot finish:



The parametric circle of radius 5

Because the standard sweep of -10 to 10 covers just over three full periods, the circle is retraced as the samples come round again; that costs nothing but plot time. Plotting is the cancellable column-by-column redraw of chapter 4, one t sample per step, and takes roughly twice as long as a single function plot because every sample evaluates both slots.

6-4 Tracing by parameter

◀ and ▶ step t one sample per press from the centre of the sweep. The readout shows the point, `X=` holding x(t) and `Y=` holding y(t); the parameter itself is never displayed, so count presses if you need to know t. With the parabola pair above, one press of ◀ shows `X=-0.078740157481` and `Y=0.0062000124001327`, the point

one t sample left of the sweep centre. The coordinate toggle on the mode page has no effect in this mode, and the free cursor still reports plain window coordinates. Trace steps take the same noticeable moment as in polar mode, so step at the calculator's pace.

The parametric table 6-5

MORE opens the chapter 4 table, and its columns fit this mode naturally: X holds the parameter (starting at 0, stepping by 1), Y1 holds x(t), Y2 holds y(t) side by side, and Y3 shows –. With the parabola pair, the Y1 column reads 0 through 5 and the Y2 column reads 0, 1, 4, 9, 16, 25. All the scrolling and step controls of chapter 4 apply.

Analysis in parametric mode 6-6

The function keys treat the *active* slot as a function of t over the window, so 2nd 1 and 2nd 2 choose whether the analyses read x(t) or y(t). One trap: after a completed replot the trace reference position sits at the sweep's end, t equal to XMAX, not at the centre. With the parabola pair and slot 2 active:

- F4 straight after a replot answers = 19.9999995, the central difference of t squared at t equal to 10;
- F5 answers = 666.66666666667, the integral of t squared over the sweep;
- 2nd F1 answers = 5.5E–21. The intersection search solves x(t) equal to y(t), a meeting of the two coordinate functions at the same t (here the numerical zero of t minus t squared), which is not a geometric self-intersection of the drawn curve.

What the mode remembers 6-7

Parametric mode keeps its own equations, enabled and active slots, window, and table position, written to the store object GPAR when you leave the mode and restored exactly when you return, just as chapter 5 describes for GPOL. The memory browser lists it as TYPE GRAPH DB, SIZE 213, and deleting it there resets the mode for its next use.

6-8 Boundaries worth knowing

Text stored in slot 3 is silently ignored by the plot and shows `UNDEF` down its table column until cleared. An axes-only plot with no error is the missing-y(t) case above. Appendix A catalogues this chapter's workflow as `parametric-editor`, `parametric-plot`, `parametric-trace`, `parametric-table`, and `parametric-analysis`.

CHAPTER

7

Differential-Equation Graphing

Differential-equation graphing plots the solution of a first-order differential equation from an initial condition, letting you watch a system evolve without finding a closed-form answer first. The mode (elsewhere `DifEq`) completes the shared graph engine of Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence). It has one fact every user must know, the frozen initial condition, documented plainly below; as throughout this book, every quoted figure was read off the machine.

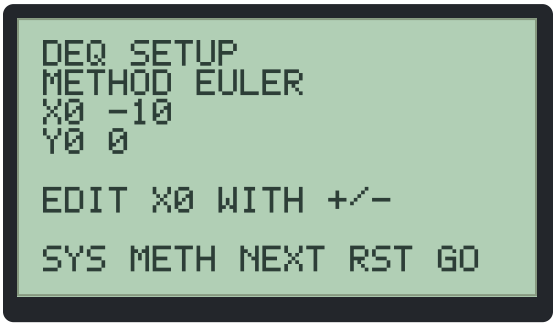
> **HARDWARE**
> differential-equation graphing and its LCD captures are validated in the emulator; physical hardware validation is reported separately.

Switching to the mode and entering an equation 7-1

Open the graph mode page as in Chapter 5 (Polar Graphing), with [2nd] [MORE] on the graph screen and then [MORE] twice, and press [F4] (`DEQ`). The middle line changes to `DIFEQ DY/DX` and the screen replots. Slot 1 holds the slope expression f in dy/dx = f(x, y), edited on the home entry line as always: [x-VAR] types the `X` and [ALPHA] [0] types the letter `Y`. Slots 2 and 3 are ignored by the plot and, as in the other modes, show `UNDEF` in the table if left holding text.

7-2 The initial condition

The solution starts from a point you choose, and both of its coordinates are settings of the mode. Press [2nd] [MORE] four times from the graph screen to reach DEQ SETUP, the fourth graph-format page:



The DEQ setup page: the method, the initial condition, and the keys that edit them

The page names the current method and shows the initial condition as X0 and Y0, and its soft keys are SYS METH NEXT RST GO. [F3] (NEXT) steps the selection from field to field, and the line above the soft keys names the one you have: EDIT X0 WITH +/−. The [+] and [-] keys move the selected value by the table's step of chapter 4, one unit by default, so two presses of [+] carry X0 from −10 to −8. [F2] (METH) cycles the method, [F4] (RST) restores the defaults, X0 at XMIN, Y0 at 0, and the method to EULER, and [F5] (GO) leaves the page and redraws. [F1] (SYS) switches between one equation and a two-state system, which the section after next covers.

Nothing has to be deleted to change a seed. The equation, the window and the table position all stay as they are, and the new initial condition takes effect on the next plot.

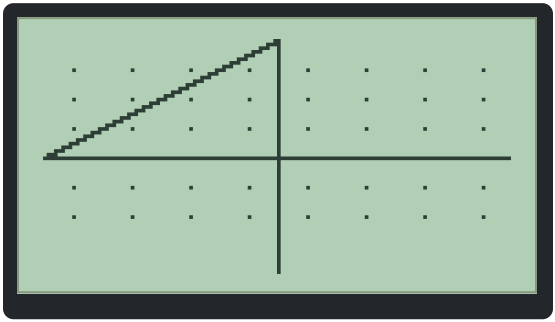
7-3 Plotting a solution

The solution starts at X0 with y at Y0 and advances in a fixed step of one 127th of the window width, one sample per plotted column. [F1] (METH) on the setup page cycles the method through EULER, HEUN and RK4. Euler takes one slope per step, Heun averages the slope at each end of the step, and RK4 combines four of them. Halving the step divides Euler's error by about two, Heun's by about four, and

RK4's by about sixteen, which is what first, second and fourth order mean in practice. The result stays deterministic: the same equation, window, initial condition and method always draw the same curve.

The step is set by the window rather than chosen adaptively, so this is not a stiff solver and does not claim to be one. An equation whose solution runs away faster than the window can follow gives up rather than inventing a curve, and reports `NO CONVERGENCE`. Other calculators pair their differential-equation modes with the differentiation-mode settings `dxDer1` and `dxNDer`; Free85 has no such settings, and the method and the window between them are the whole numerical story.

For the worked example, select the mode on a fresh machine, so the initial condition holds its defaults of `X0` at `−10` and `Y0` at `0`, press **EXIT**, type **1**, and press **GRAPH**:



The Euler solution of dy/dx=1 from the initial condition 0

A constant slope of 1 integrates to the straight line through (-10, 0) with unit slope. Three presses of **+** after **F3** (`Y0`) move the same line up through (-10, 3), with no deletion and no re-entry. For a curve the method actually shapes, plot `Y` as the equation: dy/dx = y grows an exponential across the window, and cycling **F1** through `EULER`, `HEUN` and `RK4` visibly changes where it ends up on the right. Plotting draws left to right and cancels like every other mode: **EXIT** or **CLEAR** mid-plot returns to the home screen with the equation on the entry line.

Tracing and the table

7-4

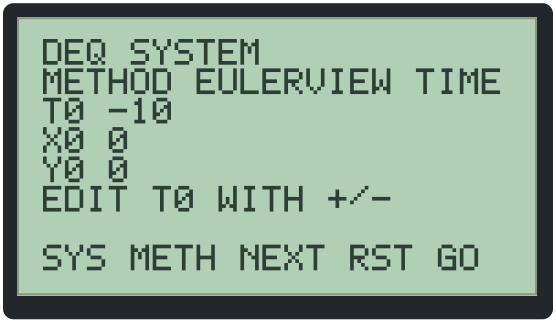
Tracing works as in chapter 4, and the readout is the integrated solution: **◀** and **▶** step the cursor along the plotted curve, and the footer reports the solution's value at that column. Each step reintegrates the solution from the left window edge, so the readout follows the cursor after a brief pause, longest near the right

edge. With the worked example's dy/dx = 1 plotted, two presses of [▶] from the centre read `X=0.393700787398` and `Y=10.393700787398`, the line y = x + 10 at that column, and the readout keeps tracking however far right you step: at `X=6.377952755878` it reports `Y=16.377952755878`.

[MORE] opens the chapter 4 table, and its `Y1` column holds the integrated solution against the ordinary `X` column: with dy/dx = 1 stored, the fresh table's `X` column reads `0` through `5` and `Y1` reads `10`, `11`, `12`, `13`, `14`, `15`, the same line row by row. The query reaches every sample of the plot, so the table reads the window from edge to edge: one press of [▼] carries `X` on to `10` with `Y1` reading `20`, and only rows past `XMAX` answer `UNDEF`.

7-5 Systems of two equations, and the phase plane

[F1] (SYS) on the setup page turns the mode from one equation into two, and the banner changes from `DEQ SETUP` to `DEQ SYSTEM`:



The DEQ system setup page, with T0, X0, Y0 and the view

In system mode the two graph slots hold the two derivatives: slot 1 is dX/dT and slot 2 is dY/dT. The variables you may name in them are `T`, `X` and `Y`, and the initial condition gains a third field, so the page lists `T0`, `X0` and `Y0`. [F3] (NEXT) steps between all three and [+] and [-] move the selected one, exactly as in the single-equation page.

`METH` still cycles `EULER`, `HEUN` and `RK4`, and in system mode both derivatives are evaluated from the same old stage before either state advances. That is what makes the pair a system rather than two independent equations: neither may see the other's new value part-way through a step.

[MORE] switches the view, which the page shows as `VIEW TIME` or `VIEW PHAS`:

- **TIME** plots the first state X against T, using the graph window as usual, and the table exposes both states.
- **PHASE** plots X against Y, with the graph window giving the bounds of the *state space* rather than of time. Its integration step is the table step, and it takes 128 samples, so the table step sets how far round an orbit the picture goes.

The worked example is the harmonic oscillator. Put Y in slot 1 and –X in slot 2, which is dX/dT = y and dY/dT = -x, set X0 to 1 with Y0 at 0, and plot. In TIME you get a cosine; in PHASE you get a circle, and the circle is the better picture, because it shows the conserved quantity that the time trace only implies.

A closed orbit is also a test of the method. Euler spirals outward on this problem, because its error always points the same way round; RK4 closes the loop to the width of a pixel. Cycling METH and replotting shows the difference more plainly than any table of numbers.

Saved state grows with the mode. A GDEQ written by an earlier firmware loads as a single equation, and is rewritten as a two-state object only when you next save.

Analysis reads the slope, not the solution

7-6

The function keys and the home-screen calculus commands stay live, but they apply chapter 3 numerics to the stored *slope expression* as a plain function of X over the window; none of them sees the plotted solution. With 1 stored, F4 answers = 0 and F5 answers = 20, the derivative and window integral of the constant slope, and F1 answers the NO NUMERIC RESULT notice because the slope never crosses zero. With X stored, EVAL(2) answers = 2, the slope at x equal to 2, not the solution's value there. To read a solution value, use the trace or the table above; the calculus keys see only the slope.

What the mode remembers

7-7

Like the other modes, differential-equation mode saves its equations, slots, window, table position, and now its initial condition and method to the store object GDEQ when you leave it, and restores them exactly when you return. Its memory-browser entry looks exactly as chapter 5 describes. A GDEQ saved by an earlier firmware still loads, and gains the setup fields on the next save; deleting the object remains a way to clear the mode entirely, but it is no longer how an initial condition is changed. Appendix A catalogues this chapter's workflow as

`diffeq-editor`, `diffeq-plot`, `diffeq-explore`, `diffeq-solve`, and `diffeq-setup`.

CHAPTER

8

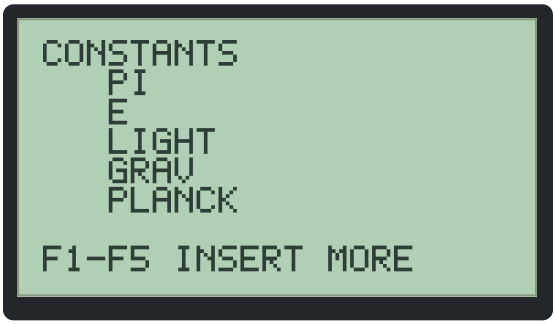
Physical and User Constants and Conversions

Two shifted keys turn the calculator into a small reference book. ***2nd 4*** *(the* `CONS` *legend) opens the constants menu, holding the mathematical and physical constants along with the constants you define yourself, and* ***2nd 5*** *(the* `CONV` *legend) opens the conversions menu, holding a pair of unit-conversion functions for each of eleven categories. Both menus work like the* `MATH` *menu of Chapter 3 (Mathematics, Calculus, and Comparisons): press the soft key under an item to insert it into your entry line and return to the home screen, or* ***MORE*** *for the next page. As everywhere else, whatever a menu inserts can also be typed letter by letter with* ***ALPHA****, or pasted from the catalog (Chapter 1: Operating the Calculator).*

The constants menu

8-1

Press 2nd 4 and the `CONSTANTS` menu lists its first page:



The constants menu opened with 2nd 4

Page one carries `PI`, `E`, `LIGHT`, `GRAV`, and `PLANCK` on F1 through F5; press MORE for `BOLTZ` and `AVOG`. A second MORE turns past them to the `USER CONSTANTS`

screen of the next section, and a third brings the first page back around. Each menu item is a plain name: the menu inserts it, and the name on its own evaluates to the stored value. Every value below is quoted from the machine:

- **PI**, the circle constant: PI answers = 3.1415926535898. It is the same PI the π legend on [2nd] [^] types.
- **E**, Euler's number: E answers = 2.718281828459. Standing alone, E is this constant; between digits it is the exponent marker typed by [EE], as chapter 3 explains, so 1E3 is a thousand, not a multiple of Euler's number.
- **LIGHT**, the speed of light in a vacuum, in metres per second: LIGHT answers = 299792458.
- **GRAV**, standard gravity, in metres per second squared: GRAV answers = 9.80665.
- **PLANCK**, the Planck constant, in joule seconds: PLANCK answers = 6.62607015E−34.
- **BOLTZ**, the Boltzmann constant, in joules per kelvin: BOLTZ answers = 1.380649E−23.
- **AVOG**, the Avogadro constant, per mole: AVOG answers = 6.02214076E23.

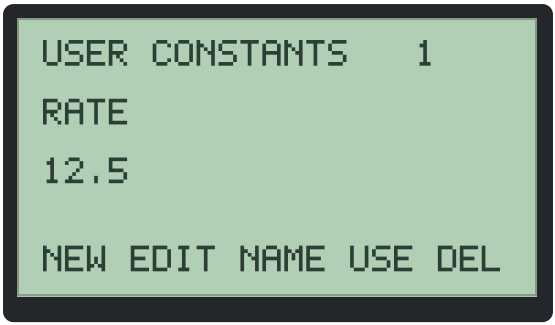
A constant behaves like any other number in an expression. 2*LIGHT answers = 599584916, and GRAV*70 answers = 686.4655, the weight in newtons of a 70 kilogram mass.

8-2 User constants

The menu's last page is not a menu but a manager. Press [MORE] twice from the first page and the USER CONSTANTS screen opens, with a count of your constants at the top right and the soft keys NEW EDIT NAME USE DEL. On a fresh machine the count is 0 and the middle of the screen says NO USER CONSTANTS.

To create a constant, press [F1] (NEW). The CONSTANT NAME prompt opens with the hint ENTER SAVE EXIT, and here, [ALPHA] latches: one press keeps letter entry on until the next press releases it, so RATE is [ALPHA] [R] [A] [T] [E]. Press [ENTER] and

the CONSTANT VALUE prompt follows; type 12.5 (pressing ALPHA first to release the letter lock) and ENTER saves the constant and returns to the screen:



The USER CONSTANTS screen holding RATE

The count now reads 1 above the name RATE and the value 12.5. A name is one to seven letters, and the value line takes a whole expression, evaluated when you save: a constant named TAU with its value typed as 2*PI is stored as 6.2831853071796.

The new name then works everywhere the built-in names do. Type RATE with ALPHA on the home screen and it answers = 12.5, exactly as GRAV answers its value.

The screen shows one constant at a time; when you hold several, ▲ and ▼ (or ◀ and ▶) step through them, MORE carries on around to the menu's first page, and EXIT returns home. The other soft keys work on the constant on show:

- **EDIT** (F2) reopens the CONSTANT VALUE prompt with an empty line: type the new value and ENTER saves it, so 8 ENTER leaves RATE reading 8.
- **NAME** (F3) opens the RENAME CONSTANT prompt for a new name; the value stays.
- **USE** (F4) drops the constant's name into the home entry line and returns home, saving the letter-by-letter typing: with RATE holding 12.5, USE and then × 2 ENTER answers = 25.
- **DEL** (F5) removes the constant on show.

A request the screen cannot honour answers the full-screen notice CONSTANT ERROR: confirming an empty name with ENTER, saving a value under a name that is not one to seven letters, or renaming onto a name already taken. CLEAR or EXIT dismisses the notice to the home screen.

Each user constant is an object in the typed store of Chapter 2 (Variables and Stored Data): the memory browser of Chapter 18 (Memory Management) lists `RATE` with `TYPE CONSTANT`, and the store's persistence carries your constants through a restart. Appendix A catalogues this workflow as `create-user-constant`, `edit-user-constant`, `name-user-constant`, and `delete-user-constant`.

8-3 The conversions menu

Press **2nd** **5** and the `CONVERSIONS` menu pages through twenty-two functions, five at a time. Each is named source-unit-first, so `INCM(` reads "inches to centimetres" and `CMIN(` the reverse, and each category contributes one such pair:

Category	Functions	Units
Length	`CMIN(`, `INCM(`	centimetres and inches
Area	`SQMFT(`, `SQFTM(`	square metres and square feet
Volume	`LGAL(`, `GALL(`	litres and US gallons
Mass	`KGLB(`, `LBKG(`	kilograms and pounds
Temperature	`CTOF(`, `FTOC(`	degrees Celsius and Fahrenheit
Time	`MINS(`, `SMIN(`	minutes and seconds
Speed	`KMHMPH(`, `MPHKMH(`	kilometres per hour and miles per hour
Pressure	`BARPSI(`, `PSIBAR(`	bar and pounds per square inch
Energy	`JCAL(`, `CALJ(`	joules and calories
Power	`WHP(`, `HPW(`	watts and horsepower
Angle	`RAD(`, `DEG(`	degrees and radians

The menu lists them in exactly this order, reading down the table row by row: page one starts with `CMIN(` and page five holds `RAD(` and `DEG(`.

A conversion is an ordinary one-argument function. To convert the boiling point of water, press **2nd** **5** **MORE** **F4** to insert `CTOF(`, then type `100)` and press **ENTER**:

CTOF(100) evaluated on the home screen

`CTOF(100)` answers `= 212`. More examples, one from each category:

- `INCM(1)` answers `= 2.54`: one inch in centimetres.
- `SQMFT(1)` answers `= 10.76391041671`: a square metre in square feet.
- `GALL(1)` answers `= 3.785411784`: a US gallon in litres.
- `KGLB(1)` answers `= 2.2046226218488`: a kilogram in pounds.
- `FTOC(32)` answers `= 0`: temperature conversions include the offset, not just a scale factor, so `CTOF(0)` answers `= 32`.
- `MINS(2)` answers `= 120` and `SMIN(90)` answers `= 1.5`.
- `KMHMPH(100)` answers `= 62.137119223733` and `MPHKMH(100)` answers `= 160.9344`.
- `BARPSI(1)` answers `= 14.503773773022` and `PSIBAR(1)` answers `= 0.068947572931678`.
- `JCAL(1)` answers `= 0.23900573613767` and `CALJ(1)` answers `= 4.184`.
- `WHP(1000)` answers `= 1.341022089595`: a kilowatt in horsepower; the reverse `HPW(1)` answers `= 745.69987158229`.
- `RAD(180)` answers `= 3.1415926535898` and `DEG(PI)` answers `= 180`, the same angle converters chapter 3 uses alongside the `RAD/DEG` angle mode.

Because the arithmetic is the fourteen-digit decimal of chapter 3, a conversion and its inverse can fall just short of an exact round trip: `CMIN(2.54)` answers `= 0.99999999999999`, not `= 1`. The residue is the true product of the stored

conversion factors, and it is why a chain of conversions is best done in one expression rather than by retyping rounded intermediate results.

Conversions nest and combine like any function: `CTOF(FTOC(32))` answers `= 32`, and `KMHMPH(3.6)` answers `= 2.2369362920544`, converting a metre per second expressed as `3.6` kilometres per hour.

CHAPTER

9

Strings and Characters

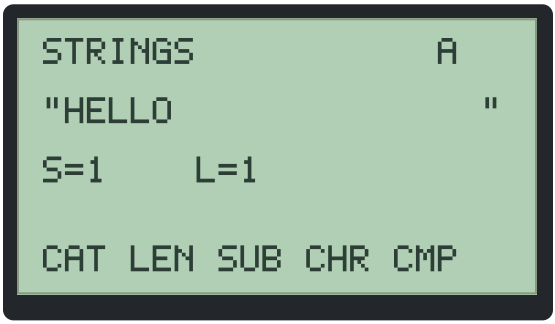
Strings are text values: names, labels, units, and messages. Free85 keeps its string tools together on one screen, the strings editor, where you build text in two working registers and apply the operations from the soft keys. This chapter covers that editor, its ten operations, and the character palette that supplies the punctuation the keyboard does not carry.

One boundary up front: strings live in the editor, not in the expression language. A quoted literal on the home entry line, such as `"A"`, answers `SYNTAX ERROR`, and no string ever becomes a named variable: the typed object store of Chapter 2 (Variables and Stored Data) keeps a string type reserved. By the same design that keeps the other editors' working values out of the memory browser of Chapter 18 (Memory Management), your text lives in the three registers described below.

The strings editor

9-1

Press 2nd 6 (the `STRNG` legend) to open the editor:



The strings editor with HELLO in register A

Reading from the top: the banner STRINGS names the screen, and the letter at the top right names the register you are looking at. There are three: A and B, the two working strings, and R, where operations leave a string result. [x-VAR] cycles the view through A, B, R, and back. Below the banner, the register's text sits between two " marks; the quotes are drawn by the screen, not characters in the string. The middle of the screen carries two counters, S= and L=, used by the substring operations described below. The soft keys hold the operations: CAT LEN SUB CHR CMP on the first page and, after [MORE], N2S S2N CPY SWP CLR.

[EXIT] returns to the home screen. The registers keep their contents: leave, calculate something, and press [2nd] [6] again, and your text is still there.

9-2 Typing into a string

Typing edits the register on show, appending at the end:

- **Letters** go through [ALPHA], one letter at a time: press [ALPHA] and the indicator ALPHA appears above the soft keys, then press a letter key to append that letter. Pressing [ALPHA] twice disarms it. So HELLO is [ALPHA] [H], [ALPHA] [E], [ALPHA] [L], [ALPHA] [L], [ALPHA] [O], using the blue letter printed beside each key.
- **Digits and common symbols** need no prefix: [0] through [9], [.], [+], [×], [÷], and [,] append 0 to 9, ., +, *, /, and , directly, and [(-)] appends −.
- **Everything else** comes from the character palette, described at the end of this chapter.

[DEL] removes the last character, and [CLEAR] empties the register on show. A register holds up to 31 characters; a 32nd answers the notice STRING TOO LONG. If the result register R is on show when you type, the editor hops back to A and appends there, so the two editable registers are always A and B.

9-3 The string operations

The first soft-key page works on registers A and B. With HELLO typed into A and WORLD into B (type the first, press [x-VAR], type the second), every result below is quoted from the machine:

- **CAT** ([F1]) concatenates A then B into R (elsewhere Concatenate): the view switches to R showing "HELLOWORLD (this book quotes the opening mark only;

the closing " stays at the right edge of the screen). The combined length must also fit in 31 characters, or the answer is the STRING TOO LONG notice.

- **LEN** (F2) measures A (elsewhere lngth): with HELLO in A it shows 5 beneath the counters.
- **SUB** (F3) copies a substring of A into R (elsewhere sub), starting at position S= and taking L= characters. ◀ and ▶ lower and raise S, ▲ and ▼ raise and lower L, both counting from 1. With HELLO in A, set S=2 and L=3 (press ▶ once and ▲ twice) and SUB shows "ELL. The copy stops at the end of the string: with HE in A, S=2, and L=5, SUB shows "E.
- **CHR** (F4) extracts the single character at position S=: with HELLO in A and S=2, CHR shows "E. It is a one-character substring, and it resets L= to 1.
- **CMP** (F5) compares A with B character by character and answers a number: −1 when A sorts earlier, 0 when the two are identical, and 1 when A sorts later. APPLE against BANANA answers −1.

LEN, SUB, and CHR always read register A, whichever register is on show; to measure B, swap it into A first with SWP below.

Numbers in and out of strings

9-4

The second soft-key page (MORE) converts between strings and numbers and manages the registers:

- **N2S** (F1) formats the home screen's most recent answer into R. Evaluate 123.5 on the home screen, press 2nd 6 MORE F1, and R shows "123.5.
- **S2N** (F2) parses A as a number. With −12.5 in A (typed with (-), the digits, and .), S2N shows −12.5, and the value becomes the home screen's ANS: press EXIT, then 2nd (-) + 1 ENTER, and ANS+1 answers = −11.5. Text that does not parse, such as HELLO, answers the notice INVALID NUMBER.
- **CPY** (F3) copies A over B and shows B.
- **SWP** (F4) swaps A and B.
- **CLR** (F5) empties the register on show.

Only S2N feeds a value back to the home screen; the numeric results of LEN and CMP are shown in the editor but leave ANS untouched.

9-5 The character palette

The palette, opened with 2nd 0 and introduced in chapter 1, is the route to everything the keyboard does not type: the space, the twenty-five punctuation marks from ! to _, and after them the Greek and international characters, fifty-four in all. Chapter 1 walks the full set; what matters here is that the palette remembers where you came from. Opened from the home screen it inserts into the entry line, and opened from the strings editor it appends to the active string and returns to the editor. So from the strings editor, 2nd 0 ▶ ENTER appends ! to the string, and a five-character HELLO becomes HELLO! with LEN showing 6. The extended characters append exactly the same way: twenty-six steps right of the space character is the capital Alpha, and ENTER there leaves it in the register, a one-character string by LEN's count.

9-6 Strings and equations

Some calculators convert between equations and strings, so that a stored function can be edited as text and text can become a function; their manuals call the two directions Eq→St and St→Eq. Free85 carries the round trip as a pair of program instructions: STTOEQ A,1 writes string register A into graph equation slot 1 of Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence) and enables it for plotting, and EQTOST 1,B reads a slot back into register B. The pair lives in the program language only, with no home-screen or soft-key route, so Chapter 16 (Calculator Programming) is its home; this chapter only records that the trip is exact. A program that runs STTOEQ A,1 and then EQTOST 1,B with 2X+1 in register A ends with register B showing "2X+1 in the strings editor.

CHAPTER

10

Number Bases and Boolean Operations

Everyday arithmetic in Free85 is the fourteen-digit decimal of Chapter 3 (Mathematics, Calculus, and Comparisons). For work with bits, this chapter's tools treat a whole number as a 16-bit machine word instead: the number-base screen displays a result in decimal, hexadecimal, octal, or binary; prefixed literals let you type a number in any of those bases; and eight Boolean functions operate on the word's bits. Every example below was run on a fresh machine, and every result is quoted exactly as the calculator displays it.

The signed word model 10-1

A 16-bit word holds the whole numbers from `−32768` through `32767`, stored as two's complement: the bit patterns `0x0000` through `0x7FFF` are `0` through `32767`, and `0x8000` through `0xFFFF` continue as `−32768` through `−1`. So all sixteen bits set, `0xFFFF`, is not 65535 but `−1`, and the pattern with only the top bit set, `0x8000`, is `−32768`.

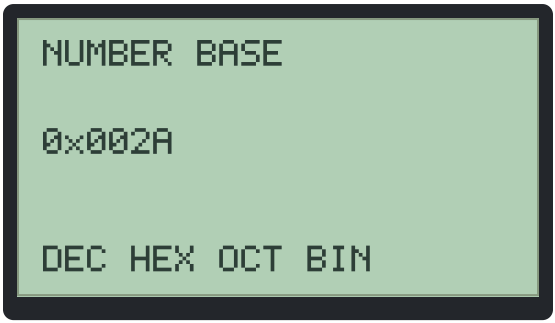
The word model applies exactly where this chapter says it does: the number-base screen, the range of prefixed literals, and the Boolean functions. Ordinary arithmetic is untouched and keeps its full decimal range, as the last section shows.

The number-base screen 10-2

Type 42 **ENTER** on the home screen, then press **2nd** **1** (the `BASE` legend). The `NUMBER BASE` screen opens with the soft keys `DEC HEX OCT BIN` on **F1** through **F4**, and each shows the most recent answer in its base:

- **F1** (`DEC`) shows 42.

- F2 (HEX) shows 0x002A:



The number-base screen's HEX view of 42

- F3 (OCT) shows 0o000052.
- F4 (BIN) shows 0b0000000000101010.

The displays are fixed-width for the full word: four hexadecimal digits, six octal digits, and sixteen binary digits, each behind the matching prefix. Press the soft keys in any order to hop between bases, and EXIT to return home. On a fresh machine, with no answer yet, the value shown is zero: DEC shows 0 and HEX shows 0x0000.

Negative answers show their two's-complement pattern: evaluate −42 and HEX shows 0xFFD6; evaluate −1 and it shows 0xFFFF. The screen reads whichever value is the current answer, so the workflow for "what is this in binary" is simply: evaluate, 2nd 1, press a base key. These four views are Free85's counterpart of the conversion commands spelled ->Dec, ->Hex, ->Oct, and ->Bin on other calculators, and appendix A files them under the labels Dec, Hex, Oct, and Bin.

The screen insists on a value the word can hold. If the answer is fractional or out of range, any base key answers a full-screen error naming the model: 2.5 and 32768 both stop at SIGNED 16-BIT INT, with the usual CLEAR OR EXIT way back (chapter 1).

10-3 Entering numbers in other bases

A literal with a base prefix is accepted anywhere a number is accepted: 0x for hexadecimal, 0o for octal, and 0b for binary. The letters are typed with ALPHA, and case does not matter: typing the letters uppercase, as ALPHA produces them, 0X2A answers = 42, and the lowercase 0x2a answers = 42 just the same (press

2nd ALPHA to select lowercase entry, then ALPHA and a letter key as usual). This book writes the prefixes lowercase and the hexadecimal digits uppercase, matching the number-base screen.

- `0b101010` answers `= 42`.
- `0o52` answers `= 42`, and `0o777` answers `= 511`.
- `0x2A+1` answers `= 43`: the literal is a number like any other, so arithmetic carries on around it.

A prefixed literal is read as an unsigned bit pattern and lands on the word model's signed value at sixteen bits:

- `0x7FFF` answers `= 32767`.
- `0x8000` answers `= −32768`.
- `0xFFFF` answers `= −1`, so `0xFFFF+1` answers `= 0`.

Beyond sixteen bits the literal does not fit: `0x10000` answers the `NUMERIC OVERFLOW` error screen. Appendix A catalogues the four entry forms as `binary–entry`, `octal–entry`, `decimal–entry`, and `hex–entry`.

The Boolean word operations

10-4

Eight functions operate bitwise on 16-bit words. None of them sit on a menu: type the names with ALPHA, paste them from the catalog, or keep your favourites on the custom menu (chapter 1). If you are arriving from another calculator's manual, the names map like this (Free85 spelling first, then the name elsewhere): `AND(`, `OR(`, `XOR(`, and `NOT(` cover `and`, `or`, `xor`, and `not`; `SHL(` and `SHR(` cover the shifts `shftL` and `shftR`; and `ROL(` and `ROR(` cover the rotations `rotL` and `rotR`.

The logic family takes two words (`NOT(` takes one) and combines them bit by bit:

- `AND(6,3)` answers `= 2`: bitwise and of `110` and `011` is `010`.
- `OR(6,3)` answers `= 7`, and `XOR(6,3)` answers `= 5`.
- `NOT(0)` answers `= −1`: flipping all sixteen bits of zero gives `0xFFFF`, which is `−1` in the signed word. Likewise `NOT(1)` answers `= −2`. Note the contrast with the comparison operators of chapter 3, whose true-and-false results are the plain numbers `1` and `0`; `NOT(` is a bit flip, not a logical negation of those booleans.

Hexadecimal literals make the masks readable: `AND(0xFF00,0x0FF0)` answers `= 3840`, which the number-base screen's `HEX` key shows as `0x0F00`, the overlap of the two masks. In the same way, evaluate `OR(0x2A,5)` (answer `= 47`) and the `HEX` view shows `0x002F`.

The shift and rotate family takes a word and a count from 0 through 15:

- `SHL(3,2)` answers `= 12`: shifting left doubles per step, and bits pushed off the top are lost. Shift into the sign bit and the signed value goes negative: `SHL(1,15)` answers `= −32768`.
- `SHR(` is a logical shift: zeros come in at the top, whatever the sign. `SHR(−1,1)` answers `= 32767`, the pattern `0xFFFF` becoming `0x7FFF`.
- The rotations wrap around instead of losing bits: `ROL(32767,1)` answers `= −2` (`0x7FFF` becomes `0xFFFE`), and `ROR(1,1)` answers `= −32768`, the lone bottom bit wrapping to the top. A count of zero is allowed: `ROL(1,0)` answers `= 1`.

All eight functions insist on the word model. A count outside 0 through 15, a fractional operand, or an operand outside the word's range answers the `DOMAIN ERROR` screen: `ROL(1,16)`, `AND(2.5,1)`, and `AND(40000,1)` all stop there.

10-5 Words and ordinary arithmetic

The word model is a costume the number wears, not a different kind of number. A prefixed literal or a Boolean result is an ordinary value the moment it exists, and it mixes freely with the decimal arithmetic of chapter 3: `0x2A/4` answers `= 10.5`, and `SHL(3,2)+0.5` answers `= 12.5`. Only the base screen and the Boolean functions hold you to whole 16-bit words; plain arithmetic keeps the full fourteen-digit range with exponents to `127`, far beyond `32767`. When a large or fractional result then meets a word-model tool, the errors above are the boundary making itself known.

CHAPTER

11

Complex Numbers

The arithmetic of Chapter 3 (Mathematics, Calculus, and Comparisons) stays on the real line: `SQRT(-9)` on the home screen answers `DOMAIN ERROR`. For work in the complex plane Free85 provides a dedicated screen, the complex editor, where a number is held as a real and an imaginary part and the soft keys carry the operations. This chapter covers the editor, the part-reading functions, complex arithmetic, and the conversions between rectangular and polar form. Every result below is quoted from a machine booted fresh for that example.

The complex editor

11-1

Press 2nd 9 (the CPLX legend) to open the editor:

The complex editor with 3+4i in register A

Reading from the top: the banner `COMPLEX` names the screen, and the letter at the top right names the register on show. There are three, as in the strings editor of Chapter 9 (Strings and Characters): `A` and `B`, the two working numbers, and `R`, where every operation leaves its result. Pressing ALPHA switches between `A` and `B`; note that here ALPHA is a switch in its own right, not the letter prefix it is

elsewhere. The two lines RE= and IM= show the register's real and imaginary parts, so the screenshot above reads as the number 3+4i.

Entry goes part by part. The editor opens ready for the real part: type a value using the digits, [.], and [(-)] for a negative sign, and the hint line changes to EDIT followed by what you have typed so far. [DEL] removes the last character and [CLEAR] abandons the entry. Press [ENTER] and the value is stored: the first [ENTER] fills RE=, the next fills IM=, and after that the turn wraps back to RE=. The cursor keys hop between the two parts without storing anything. So 3+4i is [3] [ENTER] [4] [ENTER], and -2 alone is [(-)] [2] [ENTER].

The hint line ALPHA A/B +/− SIZE is shared with the list, matrix, and vector editors of Chapter 12 (Lists) and Chapter 13 (Matrices and Vectors). Only its first half applies here: a complex number is always exactly two parts, so [+] and [-] have nothing to resize.

[EXIT] returns to the home screen, and the registers keep their contents: leave, calculate something, press [2nd] [9] again, and your numbers are still there.

11-2 Reading a number's parts

The first soft-key page, RE IM MAG ARG CONJ, takes register A apart. With 3+4i in A ([3] [ENTER] [4] [ENTER]), each answer appears in R:

- **RE** ([F1]) extracts the real part: RE= 3 (elsewhere real).
- **IM** ([F2]) extracts the imaginary part, delivered as the real part of R: RE= 4 (elsewhere imag).
- **MAG** ([F3]) is the magnitude, the distance from the origin: RE= 5.
- **ARG** ([F4]) is the argument, the angle from the positive real axis (elsewhere angle). In the default radian mode it answers RE= 0.9272952180016; switch the mode screen of Chapter 1 (Operating the Calculator) to ANGLE DEG and the same key answers RE= 53.130102354156.
- **CONJ** ([F5]) is the conjugate, the sign of the imaginary part flipped: RE= 3, IM= −4 (elsewhere conj).

Once a result is on show the letter at the top right reads R. Press [ALPHA] to get back to an editable register and continue working.

Arithmetic with A and B

11-3

The second soft-key page (MORE) is ADD SUB MUL DIV POW. The four arithmetic keys combine A and B into R, and ENTER while R is shown copies it back into A under the ENTER USE R prompt, real and imaginary parts together. With 3+4i in A and 1+2i in B (3 ENTER 4 ENTER ALPHA 1 ENTER 2 ENTER):

- **ADD** (F1) answers RE= 4, IM= 6.
- **SUB** (F2) answers RE= 2, IM= 2.
- **MUL** (F3) answers RE= −5, IM= 10: the product (3+4i)(1+2i) with its i-squared term folded in.
- **DIV** (F4) answers RE= 2.2, IM= −0.4. Division by a zero B stops at the DIVIDE BY ZERO error screen, the same one chapter 1 shows for 1/0.
- **POW** (F5) raises A to the second power: RE= −7, IM= 24 for our 3+4i. It is the same squaring operation as SQ on the next page, and B plays no part in it.

Roots, squares, and polar form

11-4

The third soft-key page is RECT POLAR ROOT SQ CL, taken here in working order rather than key order:

- **ROOT** (F3) takes the principal square root of A. For 3+4i it answers RE= 2, IM= 1, and it is the key that finishes what the home screen refuses: put -9 in A ((-) 9 ENTER) and ROOT answers RE= 0, IM= 3, the square root of -9 that SQRT(−9) would not give.
- **SQ** (F4) squares A: 3+4i becomes RE= −7, IM= 24.
- **POLAR** (F2) converts A from rectangular to polar form (elsewhere →Pol). With 1+1i in A it answers RE= 1.4142135623731, IM= 0.78539816339746: the magnitude and the angle, a quarter of pi. After a conversion the two slots hold magnitude and angle even though their labels still read RE= and IM=, so read the pair by position.
- **RECT** (F1) converts the other way (elsewhere →Rec): treat A as magnitude and angle and answer the rectangular parts. In ANGLE DEG mode, enter magnitude 2 and angle 60 and RECT answers RE= 0.99999999999978, IM= 1.732050807569. The exact answers are 1 and the square root of 3; fourteen-digit decimal arithmetic lands a whisker away and does not pretend otherwise. Other calculators also accept a complex number typed directly in

polar form; in Free85 the route is this editor and the RECT key, and Appendix A catalogues that route as `polar-complex-entry`.

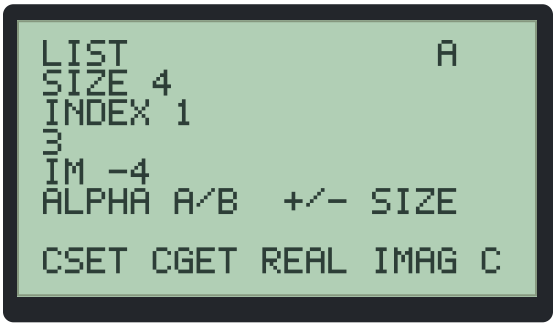
- **CL** (F5) resets all three registers to zero and shows A.

Both conversions and ARG follow the angle mode: radians in ANGLE RAD, degrees in ANGLE DEG. Other calculators also offer standing display modes that show every complex result in polar or rectangular form (elsewhere PolarC and RectC); Free85 always displays the two labelled slots, and the POLAR and RECT keys do the converting on demand.

11-5 Complex numbers elsewhere in the calculator

The expression language still has no complex literal, and a negative argument to SQRT(still answers DOMAIN ERROR rather than hopping into the complex plane on its own. The catalog of chapter 1 now lists COMPLEX, another door to this editor, alongside its other screens.

The bigger change in this release is that complex numbers travel: every collection editor of chapters 12 and 13 keeps an imaginary part for each element, set and read through a final soft-key page whose CSET key copies this editor's register A into the selected element:



A list element holding 3-4i, with its IM line

The IM -4 line under the element's value is that page's addition, and the collection arithmetic carries both parts through sums, products, and the linear algebra. Chapters 12 and 13 cover the page and the workflows; this editor remains the place where a single complex number is typed and taken apart.

CHAPTER

12

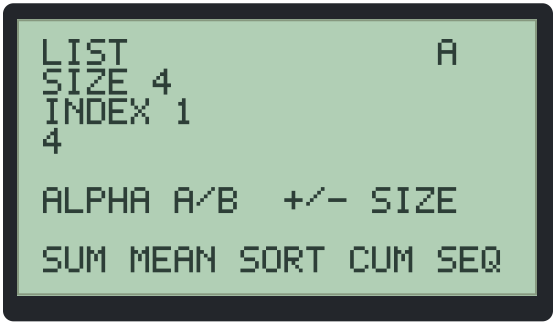
Lists

A list is an ordered collection of up to eight numbers, and Free85 keeps its list tools together on one screen, the list editor. There you build a list value by value and apply the operations from the soft keys: sums and means, sorting, running totals, sequences, element-by-element arithmetic between two lists, dimensions, fills, vector conversions, and, new in this release, imaginary parts for every element. This chapter covers the editor and all five of its soft-key pages, with every result quoted from the machine.

The list editor

12-1

Press 2nd - (the LIST legend) to open the editor. The LIST soft item on the home screen's second menu page (MORE F1, chapter 1) is another door to the same place.



The list editor holding the four-value list 4, 1, 3, 2

The banner LIST names the screen and the letter at the top right names the register on show: A and B are the two working lists and R receives every result, the same arrangement as the complex editor of Chapter 11 (Complex Numbers). ALPHA switches between A and B. Beneath the banner, SIZE is the length of the

list, INDEX is the position you are looking at, counting from 1, and the value at that position sits on the line below.

A fresh machine starts with SIZE 4. [+] lengthens the list and [-] shortens it, one value at a time, and the hint line's +/− SIZE half names the pair. The bounds are firm in this release: press [+] repeatedly and the counter stops at SIZE 8, the eight-value limit; press [-] repeatedly and it stops at SIZE 1.

Entry works one position at a time. Type a value with the digits, [.], and [(-)], and the hint line changes to EDIT followed by your typing; [DEL] removes the last character and [CLEAR] abandons the entry. [ENTER] stores the value at the current INDEX and steps forward, wrapping past the end back to position 1, so a whole list is just its values typed in order, each followed by [ENTER]. [▶] and [▼] step forward without storing, [◀] and [▲] step backward, and both wrap. To change one value, step to it, type, and press [ENTER].

[EXIT] returns to the home screen, and the registers survive the trip: reopen the editor and the values are still there.

The examples below use the list 4, 1, 3, 2, entered from a fresh machine as [2nd] [-] [4] [ENTER] [1] [ENTER] [3] [ENTER] [2] [ENTER].

12-2 Sums, sorting, and sequences

The first soft-key page is SUM MEAN SORT CUM SEQ. Each operation reads list A and leaves its answer in R:

- **SUM** ([F1]) totals the list (elsewhere sum): R becomes a single value, SIZE 1 showing 10.
- **MEAN** ([F2]) averages it: 2.5.
- **SORT** ([F3]) delivers an ascending copy (elsewhere sortA): R is a four-value list, and stepping through it with [▶] reads 1, 2, 3, 4. The descending twin, D−S, lives on the fourth page below.
- **CUM** ([F4]) answers the running totals: stepping through R reads 4, 5, 8, 10.
- **SEQ** ([F5]) ignores the values and fills R with the counting sequence 1 through SIZE (elsewhere seq): our four-value list answers 1, 2, 3, 4, and at SIZE 5 the same key answers 1 through 5.

Products, extremes, and spread

12-3

The second soft-key page (MORE) is PROD MIN MAX MED STD, again reading A into R:

- **PROD** (F1) multiplies the values together (elsewhere prod): 24.
- **MIN** (F2) and **MAX** (F3) answer the smallest and largest values: 1 and 4.
- **MED** (F4) answers the median. Our list has an even size, so the two middle values of the sorted order are averaged; the screen lingers on the sorted working copy until your next keypress, so tap an arrow key and R settles to a single value, 2.5.

  For an odd size the median is a value of the list itself, and MED leaves the whole sorted copy in R with INDEX parked on the median value's position: the three-value list 5, 1, 9 answers a three-value R with INDEX 2 showing 5.
- **STD** (F5) answers the standard deviation: 1.1180339887499. This is the population deviation, dividing by the count rather than by one less than the count.

Element-by-element arithmetic

12-4

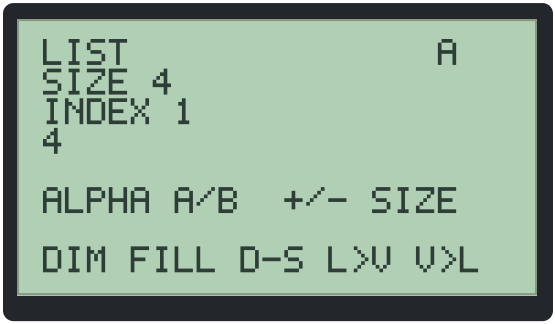
The third soft-key page is ADD SUB MUL DIV, and these combine A and B position by position. With 1, 2, 3, 4 in A and 5, 6, 7, 8 in B (type the first list, press ALPHA , type the second):

- **ADD** (F1) answers the list 6, 8, 10, 12.
- **SUB** (F2) answers −4, −4, −4, −4: every value of B is four more than its partner in A.
- **MUL** (F3) answers 5, 12, 21, 32, each pair multiplied in place, and **DIV** (F4) works the same way: the last value of its result is 0.5.

The two lists must be the same size; if they disagree, the answer is the full-screen DIMENSION ERROR notice, with the usual CLEAR OR EXIT way back (chapter 1). Dividing where B holds a zero stops at the INVALID NUMBER notice.

12-5 Dimensions, fills, and conversions

The fourth soft-key page is `DIM FILL D–S L>V V>L`:



The fourth soft-key page over the list 4, 1, 3, 2

With the chapter's 4, 1, 3, 2 in `A`:

- **`DIM`** (F1) reports the length (elsewhere `dimL`): `R` becomes `SIZE 1` holding `4`. The + and - resizing keys are the other half of the story.
- **`FILL`** (F2) fills at `A`'s length with one value, taken from the first element of `B`, just as the matrix and vector editors take scalars from `B` (chapter 13). Press ALPHA in the list editor to select `B`, type 9 ENTER, press ALPHA again if you want to watch `A`, and `FILL` answers a four-value `R` reading `9, 9, 9, 9` (elsewhere `Fill`).
- **`D–S`** (F3) is the descending sort (elsewhere `sortD`): `R` reads `4, 3, 2, 1`.
- **`L>V`** (F4) converts the list to a vector (elsewhere `li–>vc`). A vector has at most three components (chapter 13), so our four-value list stops at the `DIMENSION ERROR` notice; shorten to the three-value 4, 1, 3 with - and the same key lands you in the vector editor with its result register holding `4, 1, 3`. As in every collection editor, ENTER on the `ENTER USE R` prompt copies that result into `A` to carry it on.
- **`V>L`** (F5) converts the other way (elsewhere `vc–>li`), reading the vector editor's `A`: with a vector 7, 8, 9 stored there, `V>L` here answers the three-value list `7, 8, 9` in `R`. The same pair of keys appears in the vector editor (chapter 13).

Lists with imaginary parts

12-6

The fifth soft-key page, CSET CGET REAL IMAG CLR, gives every element an imaginary part, shown on an IM line under the element's value; the legend runs off the right edge of the screen, so CLR shows only its first letter, but F5 answers all the same. The page works element by element together with the complex editor of chapter 11:

- **CSET** (F1) copies the complex editor's register A into the selected element, both parts. Put 3-4i there (2nd 9 3 ENTER (-) 4 ENTER), come back (EXIT 2nd -), page to this page, and CSET makes element 1 read 3 with IM −4 beneath it.
- **CGET** (F2) is the reverse trip: it copies the selected element into the complex editor's result register and opens that editor on it, ready for chapter 11's operations.
- **REAL** (F3) keeps the element's real part and clears its IM to zero; **IMAG** (F4) moves the imaginary part into the value slot, so our 3-4i element becomes −4 with IM 0; **CLR** (F5) zeroes the selected element entirely.

Typing over an element the ordinary way also clears its IM to zero, so set values first and imaginary parts second. The editor pages and registers reset to the first page and A each time you re-enter, so each CSET trip is: enter the number in the complex editor, return, press MORE four times, step to the element, F1.

The arithmetic carries both parts. Build A with 2+3i and -1+2i in elements 1 and 2 (two CSET trips, elements 3 and 4 staying zero), and B with 1-2i and 3+4i (two more, pressing ALPHA after entering the editor), and the third page's ADD answers an R whose elements read 3 with IM 1 and 2 with IM 6: the sums 3+1i and 2+6i, visible by paging back to the fifth page. SUM and PROD keep both parts too: summing a list holding 1+2i and 3-1i answers 4 with IM 1.

Lists and the rest of the calculator

12-7

Lists live in this editor, not in the expression language: the home screen's entry line has no list literal, and the LIST soft item opens the editor rather than inserting anything. The statistics editor of Chapter 15 (Statistics and Statistical Plots) works on these same registers under different names: its X column is list A and its Y column is list B, so a sort or a fill here reorders the statistics too. Appendix A

catalogues the resizing keys as `->dimL`, the fill as `Fill-list`, and the position-by-position arithmetic as `elementwise-list`, with its plain-number and complex cases filed as `elementwise-real` and `elementwise-complex`. Eight values is not many, but between the sorts, the sums, the conversions, and the imaginary parts, this one screen makes them work hard.

CHAPTER

13

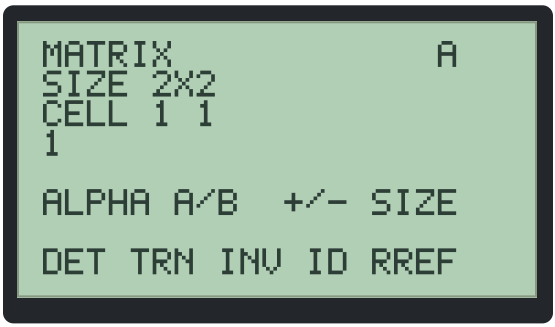
Matrices and Vectors

Matrices and vectors each have their own editor, built on the same plan as the list editor of Chapter 12 (Lists): two working registers `A` *and* `B`*, a result register* `R`*,* ***ALPHA*** *to switch between the working pair, and the operations on the soft keys. Whenever the screen is showing* `R`*, the line above the soft keys reads* `ENTER USE R`*, and pressing* ***ENTER*** *copies the whole result into* `A` *in one action, dimensions and complex components intact, so a result can be used as the next operand without being read off and typed back in. A matrix has three rows and up to six columns, and a vector has two or three components in this release. This chapter covers both editors, the linear-algebra operations from determinants to eigensystems, the coordinate conversions, and the error screens that guard them, with every result quoted from the machine.*

The matrix editor

13-1

Press 2nd 7 (the `MATRX` legend) to open the matrix editor; the `MAT` soft item on the home screen's second menu page (MORE F2, chapter 1) leads to the same place.

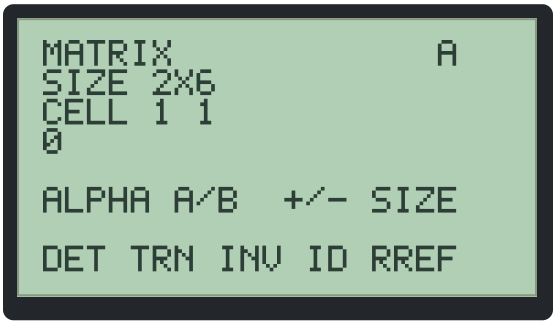


The matrix editor holding the 2 by 2 matrix 1 2 / 3 4

Under the MATRIX banner, SIZE 2X2 gives the dimensions, rows first; that is also where a fresh machine starts. [+] and [-] resize one row at a time; press [x-VAR] and the same keys resize columns instead, and [x-VAR] again hands them back to rows. Rows run from 1 to 3 and columns from 1 to 6, so pressing [+] beyond either limit changes nothing.

Those two limits are different on purpose, and the difference decides which operations are available. The editor and the A, B and R registers hold anything up to 3 by 6, real or complex, and the operations that make sense for a rectangle work across the whole of it: row operations, augmentation, RREF, addition, subtraction, scaling, and multiplication where the shapes agree. The operations that need a square keep their old 3 by 3 ceiling, because that is what they mean: DET, INV, ID, SOLVE, LU and the eigensystem.

Transposing is the case where the two limits collide. A matrix wider than three columns would transpose into more than three rows, which the workspace cannot hold, so TRN answers DIMENSION ERROR and leaves R as it was.



A matrix grown out to six columns

Six columns is what makes an augmented system comfortable rather than cramped. Three equations in three unknowns plus a right-hand side is 3 by 4; a two-product simplex tableau is 3 by 6 exactly.

The CELL line tracks the selected cell as you move: the two figures after CELL are the cell's row and column, and the value of the cell sits on the line below, so stepping through the screenshot's matrix reads CELL 1 1 to CELL 2 2 with 1, 2, 3, 4 beneath. Cells run in reading order, left to right along row 1, then row 2, and so on. Typing and storing work exactly as in the list editor: digits, [.], and [(-)] build a value on the EDIT line, [ENTER] stores it and steps to the next cell, wrapping at the end, and the cursor keys step without storing. So the matrix in the screenshot is

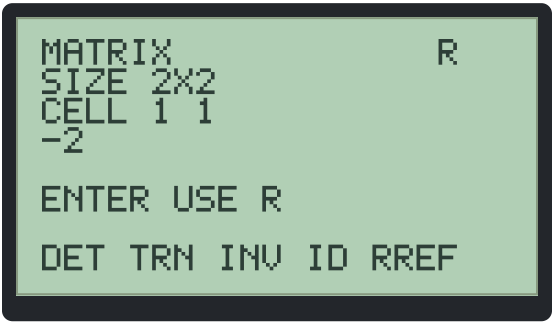
1 ENTER 2 ENTER 3 ENTER 4 ENTER from a fresh machine: row one is 1 2, row two is 3 4. EXIT leaves for the home screen and the registers keep their contents.

Matrix operations

13-2

The first soft-key page is `DET TRN INV ID RREF`, each reading `A` and answering in `R`. With the screenshot's matrix in `A`:

- **`DET`** (F1) answers the determinant as a 1 by 1 result: `R` shows `SIZE 1X1` holding `-2` (elsewhere `det`).
- **`TRN`** (F2) transposes (elsewhere `transpose`): `R` is `SIZE 2X2` and stepping through it reads `1`, `3`, `2`, `4`.
- **`INV`** (F3) inverts `A`: stepping through `R` reads `-2`, `1`, `1.5`, `-0.5`.



The inverse in the result register

- **`ID`** (F4) writes an identity matrix the size of `A` into `R` (elsewhere `Ident`): with our 2 by 2 in `A`, stepping through the result reads `1`, `0`, `0`, `1`. `A` itself is only read for its size, so its cells and the other registers are untouched.
- **`RREF`** (F5) answers the reduced row-echelon form (elsewhere `rref`), which for our invertible matrix is the identity: stepping through `R` reads `1`, `0`, `0`, `1`.

Matrix arithmetic and solving

13-3

The second soft-key page (MORE) is `ADD SUB MUL SCL SOLVE`, combining `A` and `B`. With 1, 2, 3, 4 in `A` and 5, 6, 7, 8 in `B`:

- **`ADD`** (F1) answers `6`, `8`, `10`, `12`, and **`SUB`** (F2) answers the differences, starting `-4`.
- **`MUL`** (F3) is the matrix product: `19`, `22`, `43`, `50`.

- **SCL** (F4) multiplies every cell of A by one scalar, taken from the top-left cell of B. With 2 stored there, A doubles: 2, 4, 6, 8.
- **SOLVE** (F5) solves the linear system whose coefficients are A and whose right-hand sides are the first column of B. For the system x+y=3, x-y=1, put 1, 1, 1, -1 in A and fill B as 3 ENTER 0 ENTER 1 ENTER 0 ENTER, which runs 3 and 1 down its first column and zeroes the unused second one. SOLVE answers a SIZE 2X1 result reading 2 then 1: x is 2 and y is 1. Larger simultaneous systems have a solver of their own in Chapter 14 (Equation, Polynomial, and Simultaneous Solving).

Two error screens guard the algebra, both with the usual CLEAR OR EXIT way back. Inverting a matrix with determinant zero (try 1, 2, 2, 4) stops at SINGULAR MATRIX, and combining shapes that do not fit, such as adding a 3 by 2 to a 2 by 2, stops at DIMENSION ERROR.

13-4 Row operations and augmentation

The third soft-key page is REF SWP RADD RMUL AUG, which is exactly 21 characters and so fits the screen whole. SWP is the label; the operation it runs is the full SWAP described below. The row operations work on the selected row, the row the cell cursor is sitting in, and a fresh entry wraps the cursor back to cell 1, so the examples below all start with row 1 selected. Where a second row is needed it is the following row, wrapping from the bottom back to the top, and where a scale is needed it comes from the top-left cell of B, just as SCL takes it. With the screenshot's 1, 2, 3, 4 in A:

- **REF** (F1) answers the row-echelon form (elsewhere ref). For our invertible matrix the elimination reduces all the way to the identity, so stepping through R reads 1, 0, 0, 1, the same answer as RREF.
- **SWAP** (F2, labelled SWP) swaps the selected row with the following row (elsewhere rSwap): R reads 3, 4, 1, 2.
- **RADD** (F3) adds the following row, scaled by B's top-left cell, to the selected row. With 2 stored there (ALPHA 2 ENTER), row 1 gains twice row 2 and R reads 7, 10, 3, 4. Other calculators split this into an unscaled and a scaled form (elsewhere rAdd and mRAdd); Free85's one key covers both, with a scale of 1 for the plain sum.
- **RMUL** (F4) multiplies the selected row by the same scale (elsewhere multR): with 2 in B, R reads 2, 4, 3, 4.

- **AUG** (F5) appends B's columns to A (elsewhere aug). Shrink B to a 2 by 1 column (ALPHA x-VAR -) holding 5, 6, and AUG answers a SIZE 2X3 result whose rows read 1, 2, 5 and 3, 4, 6.

Norms, condition, and random fills

13-5

The fourth soft-key page is NORM RNORM CNORM COND RND. This legend overruns the screen by a full key: only the first four names fit, and the fifth, RND, sits beyond the right edge entirely, but F5 still answers. With 1, 2, 3, 4 in A, each answer is a SIZE 1X1 result:

- **NORM** (F1) is the Frobenius norm, the square root of the sum of the squared cells: 5.4772255750515, fourteen-digit arithmetic's take on the square root of 30.
- **RNORM** (F2) is the largest row sum of absolute values (elsewhere rnorm): 7, from the row 3, 4.
- **CNORM** (F3) is the largest column sum (elsewhere cnorm): 6, from the column 2, 4.
- **COND** (F4) is the condition number (elsewhere cond), the Frobenius norm of A times the Frobenius norm of its inverse: 15. A singular A has no inverse and no condition number, so the key stops at the same SINGULAR MATRIX notice as INV (try 1, 2, 2, 4).
- **RND** (F5) fills every cell at A's size with values from the same deterministic sequence as chapter 3's RAND (elsewhere randM): from a fresh boot a 2 by 2 answers 0.7968, 0.8984, 0.4492, 0.7246 in reading order.

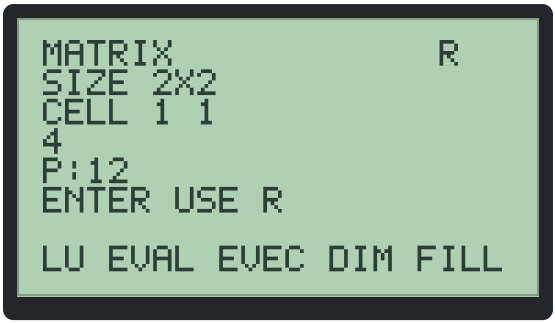
Decompositions and eigensystems

13-6

The fifth soft-key page is LU EVAL EVEC DIM FILL:

- **LU** (F1) factorises A into one combined result: the upper triangle U on and above the diagonal, and the multipliers of a unit-diagonal L below it. For 4, 3,

6, 3 the result is



The combined LU factors of 4 3 / 6 3

and stepping through `R` reads `4`, `3`, `1.5`, `–1.5`: `U` is the rows 4, 3 and 0, -1.5, and 1.5 is the multiplier that rebuilds row 2 as 1.5 times row 1 plus 0, -1.5. The factorisation pivots when it must: a zero leading cell (try 0, 1, 2, 3) swaps the rows first and answers `2`, `3`, `0`, `1`. The row order is printed on the result screen itself, as `P:` followed by one digit per row: the unpivoted 4, 3, 6, 3 above reads `P:12`, and this one reads `P:21`, the original rows in the order the factorisation used them. Vector `R` is not touched by any of this, real or imaginary parts.

- **`EVAL`** (F2) answers the eigenvalues (elsewhere `eigVl`) as a `SIZE 1X2` (or `1X3`) row. The symmetric 2, 1, 1, 2 answers `3` then `1`. Complex pairs use the imaginary plane of the final page: the rotation matrix 0, -1, 1, 0 answers two cells reading `0`, and paging to the final soft-key page shows `IM –1` and `IM 1` beneath them. A 3 by 3's roots take the machine far longer to find than a 2 by 2's, so give it time.
- **`EVEC`** (F3) answers the matching eigenvectors (elsewhere `eigVc`), normalised to length one and stored one per column. For 2, 1, 1, 2 the result reads `0.70710678118655`, `0.70710678118655`, `0.70710678118655`, `–0.70710678118655`: the first column is the eigenvector for 3, the second for 1, each a scaled 1, 1 or 1, -1.
- **`DIM`** (F4) reports the dimensions: a `SIZE 1X2` result reading `2` then `2` for our square `A`.
- **`FILL`** (F5) fills at `A`'s size with the value in `B`'s top-left cell: with 9 stored there, `R` reads `9`, `9`, `9`, `9`.

Matrices with imaginary parts

13-7

The sixth and final soft-key page, `CSET CGET REAL IMAG CLR`, is the same imaginary-parts page the list editor carries, worked element by element with the complex editor: chapter 12 walks through it. An `IM` line under the cell's value shows the selected cell's imaginary part, `CSET` copies the complex editor's `A` into the cell, and typing over a cell the ordinary way clears its `IM` to zero. Addition, subtraction, `SCL`, and the matrix product `MUL` all carry both parts: with 1+1i in `A`'s top-left cell and 2+3i in `B`'s, `ADD` answers a top-left cell of `3` with `IM 4`.

The vector editor

13-8

Press 2nd 8 (the `VECTR` legend), or MORE F3 (`VEC`) from the home screen, to open the vector editor:

The vector editor holding the vector 3, 4, 0

A vector is a single column of components: `SIZE 3` on a fresh machine, `COMP` naming the component on show, and the same entry rules as the other editors. + and - switch the length between 2 and 3, the two sizes this release supports, and the second soft-key page offers the same choice via its `2D` and `3D` keys. The `RECTV` tag beside the size names the coordinate form on show, rectangular to begin with; the conversions below can change it. The vector above is 3 ENTER 4 ENTER 0 ENTER.

Vector operations

13-9

The first soft-key page is `MAG NRM DOT CRS ANG`. With 3, 4, 0 in `A` and 1, 2, 3 in `B`:

- **MAG** (F1) answers the magnitude of `A` (elsewhere `norm`): `5`.

- **NRM** (F2) normalises A to length one (elsewhere unitV): stepping through R reads 0.6, 0.8, 0.
- **DOT** (F3) answers the dot product with B (elsewhere dot): 11.
- **CRS** (F4) answers the cross product with B (elsewhere cross): 12, −9, 2.
- **ANG** (F5) answers the angle between A and B, following the angle mode of chapter 1: 0.9422435660893 in ANGLE RAD, and 53.986579610272 with the mode set to ANGLE DEG.

The second page (MORE) is ADD SUB SCL 2D 3D:

- **ADD** (F1) answers 4, 6, 3.
- **SUB** (F2) answers the differences, ending −3.
- **SCL** (F3) multiplies A by the first component of B, just as the matrix SCL uses B's top-left cell.
- **2D** (F4) and **3D** (F5) set the length, as above.

Normalising a vector of zeros stops at the ZERO VECTOR notice, and CRS insists on three components: with two-component vectors it answers DIMENSION ERROR, since the cross product only lives in three dimensions.

13-10 Coordinate conversions

The third soft-key page, R>CY CY>R R>SP SP>R, converts a three-component A between rectangular, cylindrical, and spherical coordinates, and each conversion sets the tag beside SIZE. The angle components follow the angle mode of chapter 1's mode screen, just as ARG does in the complex editor of Chapter 11 (Complex Numbers): in ANGLE DEG mode the first example below answers its angle as 53.130102354156 instead. The worked figures in this section all assume the fresh machine's ANGLE RAD. With 3, 4, 0 in A:

- **R>CY** (F1) converts rectangular to cylindrical (elsewhere –>Cyl): R carries the CYLV tag (elsewhere the CylV display mode) and reads 5, 0.9272952180016, 0: the distance from the vertical axis, the angle around it, and the height.
- **CY>R** (F2) reads A as a cylindrical triple and converts it back to rectangular, restoring the RECTV tag (elsewhere RectV). Enter 2, 1.5707963, 7 and it answers 5.34594384493E−8, 1.9999999999882, 7: a right angle in fourteen digits lands just off the axis, as chapter 11's polar conversions do.

- **R>SP** (F3) converts rectangular to spherical (elsewhere –>Sph): the tag becomes SPHEREV (elsewhere SphereV) and our 3, 4, 0 answers 5, 0.9272952180016, 1.5707963267949: the distance from the origin, the same angle around the vertical axis, and the angle down from it, a right angle for a vector in the horizontal plane.
- **SP>R** (F4) reads A as a spherical triple and converts back: 2, 0, 1.5707963 answers 1.9999999999882, 0, 5.34594384493E–8.

The conversions read A even while the screen shows R, so pressing CY>R straight after R>CY does not undo the trip: it reads the rectangular 3, 4, 0 still in A as if it were cylindrical. To chain conversions, press ENTER on the ENTER USE R prompt first: that moves the result into A, and the next conversion reads what you just computed. The tag names the last conversion's form rather than tracking each register: it survives leaving and re-entering the editor, stays put when a list conversion delivers a plainly rectangular vector, and only CY>R and SP>R switch it back to RECTV, so treat it as a note of where the conversions last left off.

Vector dimensions, fills, and conversions

13-11

The fourth soft-key page is DIM FILL NORM V>L L>V, the vector editor's own copy of the list editor's fourth page (chapter 12). With 3, 4, 0 in A:

- **DIM** (F1) reports the length: R becomes SIZE 1 holding 3.
- **FILL** (F2) fills at A's length from B's first component: with 6 stored there, R reads 6, 6, 6.
- **NORM** (F3) answers the Euclidean length, 5 for our vector, the same figure as the first page's MAG.
- **V>L** (F4) hands the vector to the list editor (elsewhere vc–>li): its result register receives 3, 4, 0. **L>V** (F5) is the return trip for lists of at most three values (elsewhere li–>vc), as chapter 12 shows.

The fifth soft-key page is the imaginary-parts page CSET CGET REAL IMAG CLR, element by element as in the other editors (chapter 12); DOT and CRS carry both parts through their products.

Appendix A catalogues the matrix inversion, Frobenius norm, dimension report, resizing, and fill as inverse–matrix, norm–matrix, dim–matrix, –>dimM, and Fill–matrix; their vector counterparts as dim–vector, –>dimV, Fill–vector, and

`norm-vector`; and the position-by-position arithmetic of both editors as `elementwise-matrix` and `elementwise-vector`.

CHAPTER

14

Equation, Polynomial, and Simultaneous Solving

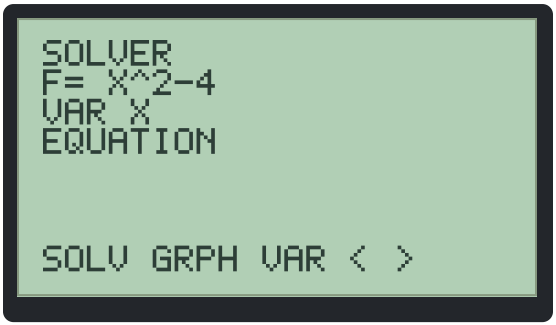
Free85 has three solving tools. The general solver keeps an equation in any single letter and hunts for a value that makes it zero. The polynomial editor takes the coefficients of a polynomial of degree 2 to 4 and answers every root, real or complex. The simultaneous editor takes a linear system of up to four equations and answers the unknowns, or tells you why it cannot. The first is a workspace behind ***2nd GRAPH****; the other two are editors built on the same entry rules as the collection editors of Chapter 13 (Matrices and Vectors), whose* `SOLVE` *soft key is a fourth route to small linear systems. Every figure in this chapter is quoted from the machine.*

The general solver

14-1

The GRAPH key's shifted function is `SOLVER` (elsewhere `Solver`), a persistent workspace rather than a one-shot command. Press 2nd GRAPH and the screen changes to the `SOLVER` banner, an `F=` line naming the stored equation, a `VAR X` line naming the unknown, a field area, and the soft keys `SOLV GRPH VAR < >`. On a

fresh machine nothing is stored yet, so the F= line shows the <HOME EXPRESSION> placeholder.



The solver workspace holding X^2-4

The equation arrives from the home screen. Whatever is on the home entry line when you press [2nd] [GRAPH] becomes the stored equation, so [x-VAR] [x²] [-] [4] [2nd] [GRAPH] opens the workspace with F= X^2−4, as the screenshot shows. Entering with an empty entry line keeps what is already stored, and the equation, unknown, guess, and bounds all survive [EXIT] and a later reopen intact.

VAR ([F3]) steps the unknown through the letters, X to Y to Z to A and on around the alphabet, so three presses turn VAR X into VAR A for an equation written in A. The < and > keys ([F4] and [F5]) page the field area through EQUATION, VARIABLE, GUESS, LOWER, and UPPER; a fresh machine holds guess 0 with bounds −10 and 10. On a numeric page, digits, [.], and [(-)] build a value on an EDIT line, [ENTER] stores it and pages onward, and [CLEAR] abandons the half-typed line, following the entry rules of the editors below. So [(-)] [5] [ENTER] on the LOWER page stores −5 and pages on to UPPER, and paging back shows LOWER still holding −5.

SOLV ([F1]) hunts for a root between the bounds and publishes a ROOT line and a RES residual line in the field area. The guess is tried first, so a guess that already solves the equation comes back exact: store A^2−9, turn VAR X into VAR A, page to the guess and store −3, and SOLV answers a ROOT of −3 with RES 0. A guess of 2 on X^2−4 answers a ROOT of 2 the same way, which is how you pick between roots. Any other guess sends the solver scanning 32 subintervals of the bounds for a sign change, then bisecting that subinterval, up to 40 halvings, until the residual passes the numeric tolerance set with [2nd] [CLEAR], the TOLER legend of Chapter 3 (Mathematics, Calculus, and Comparisons): X^2−4 from the default guess answers

a ROOT of –1.9999998807909 with RES–4.768364E–7, the -2 crossing under a little numerical dust, its residual inside the fresh 1E–6 tolerance.

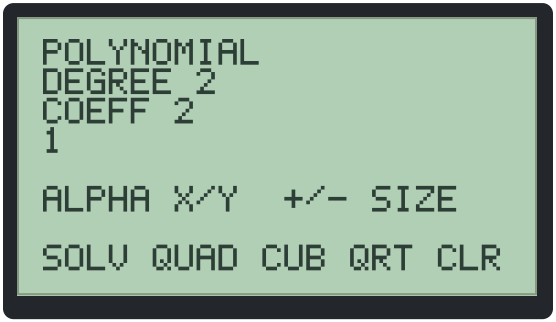
The bounds pick between roots just as the guess does, by fencing the scan in. Page to the bounds and store 1 and 5, and SOLV answers a ROOT of 2 with RES 0, the positive crossing alone inside the interval; store –5 and 0 with a guess of –3 instead and it answers the other root, a ROOT of –1.9999998807909 with RES –4.768364E–7. A guess outside the bounds does no harm: it is still tried first, and the scan then keeps to the bounded interval.

Four notices guard the search, each with the usual CLEAR OR EXIT way back. SOLV with no stored equation stops at ENTER EQUATION HOME. Bounds out of order stop at LOWER MUST BE < UPP (store –20 as the upper bound and try); the screen clips the last letters, short for lower must be less than upper. An equation that cannot be evaluated across the bounds, such as LN(X)–1 over the default –10 to 10, stops at EQUATION DOMAIN ERR, clipped the same way from equation domain error. And an equation with no sign change between the bounds, such as X^2+1, stops at NO BOUNDED ROOT.

GRPH (F2) hands the problem to the graph screen of Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence). The stored equation becomes the active graph equation with the solver's unknown renamed to the graph variable, so TAN(A)–1 hands off as TAN(X)–1; the solver bounds become the window's horizontal range; and the plot opens, where the graph screen's own F1 root finder and its companions (chapter 4) take aim at whichever crossing you can see. Appendix A catalogues this workspace as solver–equation, solver–variables, solver–guesses, solver–bounds, and solver–graph.

14-2 The polynomial editor

Press [2nd] [PRGM] (the POLY legend, elsewhere poly) to open the polynomial editor:



The polynomial editor holding x^2-5x+6

Under the POLYNOMIAL banner, DEGREE 2 names the degree, and the COEFF line tracks the selected coefficient by its power: COEFF 2 is the coefficient of x^2 and COEFF 0 is the constant, with the selected coefficient's value on the line below. A fresh machine holds the polynomial x^2, a leading 1 with zeros behind it.

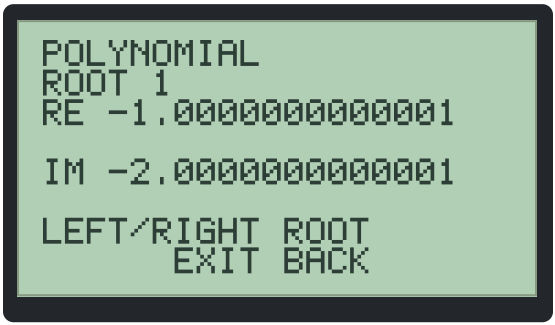
QUAD ([F2]), CUB ([F3]), and QRT ([F4]) set the degree to 2, 3, or 4, and [+] and [-] step it one at a time between the same limits; degree 4 is the release ceiling, so pressing [+] beyond DEGREE 4 changes nothing. Entry follows the editors of chapter 13: digits, [.], and [(-)] build a value on the EDIT line, [ENTER] stores it and steps to the next lower power, wrapping past the constant, and the cursor keys step without storing. [CLEAR] abandons a half-typed EDIT line, CLR ([F5]) resets every coefficient to the fresh polynomial, and [EXIT] leaves for the home screen with the coefficients kept. An entry that does not parse as a number (a bare [.], say) stops at the INVALID NUMBER notice. So the screenshot's x^2-5x+6 is [1] [ENTER] [(-)] [5] [ENTER] [6] [ENTER] from a fresh editor, the display wrapping back to COEFF 2 when the constant is stored.

14-3 Reading the roots

SOLV ([F1]) computes the roots and replaces the coefficient display with a root browser: ROOT 1 names the root on show, RE and IM give its real and imaginary parts, and [◀] and [▶] step through the roots, as the LEFT/RIGHT ROOT hint says.

For x^2-5x+6 the browser opens on ROOT 1 with RE 3 and IM 0, and ▶ shows ROOT 2 with RE 2 and IM 0: the roots 3 and 2, as they should be. Some inputs leave a little numerical dust from the iterative search in the last digit or two of a root; treat it as the nearest round value. A value too long for the 21-character line clips at the right edge. CLEAR steps back to the editor with the coefficients kept, and EXIT leaves for the home screen.

Complex roots come out the same way. Solve x^2+2x+5 (coefficients 1, 2, 5) and ROOT 1 reads RE −1.0000000000001 with IM −2.0000000000001, while ROOT 2 reads RE −1 with IM 2: the conjugate pair -1±2i, the first root carrying that dust in its last digit.



The roots of x^2+2x+5

The higher degrees work the same. The cubic x^3-6x^2+11x-6 (press CUB, then coefficients 1, -6, 11, -6) answers RE 3, RE 1, and RE 2.0000000000016 across its three roots, each with IM 0. The quartic x^4-5x^2+4 (press QRT, then 1, 0, -5, 0, 4) answers RE 2, RE −1, RE −2, and RE 1. Solving with a zero leading coefficient stops at the LEADING COEFF ZERO notice, since the polynomial would really be one of lower degree.

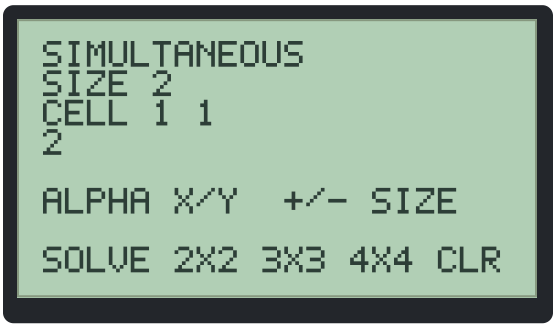
Quadratics with mixed-sign roots

Earlier firmware misconverged on any quadratic whose two real roots differ in sign, and this guide once taught a degree-3 workaround for them. This release repairs the degree-2 search, so such quadratics solve directly. x^2-x-6 (coefficients 1, -1, -6) answers ROOT 1 with RE 3 and IM 0, then ROOT 2 with RE −2 and IM 0: the roots 3 and -2 exactly. x^2-4 (1, 0, -4) answers RE 2.0000000000001 and RE −2.0000000000001, the roots 2 and -2 under a grain of dust, and x^2-6x+8 (1, -6, 8), which once stalled short of its roots, answers RE 4 and RE 2. A

negative leading coefficient, which once upset the search at every degree, is also safe now: -x^2+4 (coefficients -1, 0, 4) answers the same values as x^2-4, so there is no need to multiply an equation through by -1 before solving. The old workaround still works if you meet it in earlier notes: CUB with coefficients 1, -1, -6, 0 multiplies x^2-x-6 by x, and the browser answers RE 3, RE −2, and RE 0, the true roots plus the 0 the extra factor added; or aim the general solver at each root with a guess, as the workspace section above shows. The cubic and quartic searches answered every polynomial we put to them, at worst with a small residue in the last digits.

14-4 The simultaneous editor

Press 2nd STAT (the SIMULT legend, elsewhere simult) to open the simultaneous-equation editor:

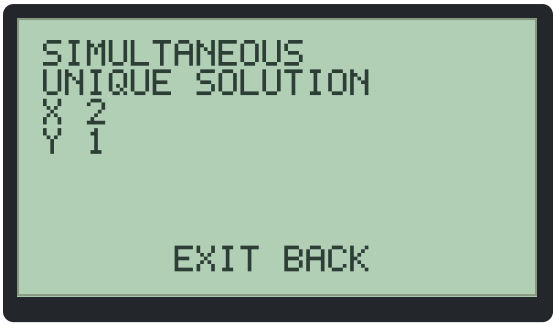


The simultaneous editor holding a 2 by 2 system

Under the SIMULTANEOUS banner, SIZE 2 gives the number of equations. 2X2 (F2), 3X3 (F3), and 4X4 (F4) set the size, + and - step it, and 4 is the release ceiling. The CELL line reads row then column, both counted from one, and the column follows the cursor along the row: stepping once to the right from CELL 1 1 reads CELL 1 2. The right-hand side is the last column of the row, so a 2 by 2 system's right-hand sides sit at CELL 1 3 and CELL 2 3. Cells run row by row: each row takes its coefficients left to right and then its right-hand side, and ENTER steps through them with the same entry rules as the polynomial editor. CLR (F5) zeroes every cell, and EXIT keeps the contents.

The screenshot's system is 2x+y=5 and x-y=1: from a fresh editor press 2 ENTER 1 ENTER 5 ENTER 1 ENTER (-) 1 ENTER 1 ENTER. SOLVE (F1)

answers a result screen reading `UNIQUE SOLUTION` with `X 2` and `Y 1`: x is 2 and y is 1.



The unique solution of the 2 by 2 system

The result screen answers only to [EXIT], which leaves for the home screen; to change the system, press [2nd] [STAT] again and the editor reopens with every cell kept. A 3 by 3 example: enter 2x+y-z=8, -3x-y+2z=-11, and -2x+y+2z=-3 row by row and `SOLVE` answers `X 2`, `Y 3`, and `Z −1`. At 4 by 4 the fourth unknown's label renders as `[`, the character after `Z` in the character set, so a system whose solution is 1, 2, 3, 4 reads `X 1`, `Y 2`, `Z 3`, and `[ 4`.

Singular systems

14-5

A system without a unique solution gets a status screen instead of numbers, and both states are recoverable with [EXIT]. Contradictory equations (enter x+y=1 and x+y=2) answer `NO SOLUTION`, and dependent equations (enter x+y=2 and 2x+2y=4) answer `UNDERDETERMINED`, meaning a whole family of solutions fits. This mirrors the `SINGULAR MATRIX` guard on the matrix editor's inverse (chapter 13), but here the two degenerate cases are told apart.

CHAPTER

15

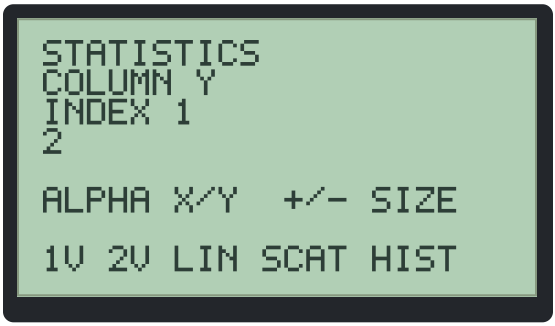
Statistics and Statistical Plots

The statistics editor holds two columns of data, computes one-variable and two-variable summaries, fits seven regression models, forecasts from the fitted model in either direction, and draws four kinds of plot. Every formula states whether it uses the sample or the population definition, and every figure in this chapter is quoted from the machine.

The statistics editor

15-1

Press STAT to open the statistics editor; the STAT soft item on the home screen's second menu page (MORE F4, chapter 1) leads to the same place.



The statistics editor holding paired data

Under the STATISTICS banner, COLUMN X names the column you are in, and ALPHA switches between the X and Y columns; one-variable work uses X alone, and paired data puts the second coordinate in Y. The INDEX line counts the entries, and the value of the selected entry sits on the line below. Both columns share one length: a fresh machine holds four entries, + and - grow and shrink the pair together, and eight is the ceiling. The editor never prints the length, so

watch INDEX: ENTER wraps back to INDEX 1 after the last entry, and that wrap tells you where the end is.

The columns are not a private store: X is list A and Y is list B of Chapter 12 (Lists), the same registers under a different banner. A 9 stored at INDEX 1 of the list editor's A greets you as the first X entry here, and everything the list editor does to A and B, the sorts and fills included, lands in the statistics columns.

Entry follows the collection editors of chapter 12: digits, . , and (-) build a value on the EDIT line, ENTER stores it and steps to the next entry, and the cursor keys step without storing, wrapping in both directions. To correct an entry, step back to it and type the replacement. CLEAR abandons a half-typed EDIT line, and an entry that does not parse as a number (a bare . , say) stops at the INVALID NUMBER notice. There is no key that clears the data: shrink the length with - or overwrite the cells. EXIT leaves for the home screen and the columns keep their contents.

Six pages of soft keys cycle under MORE: 1V 2V LIN SCAT HIST, then MEAN MED VAR SSD PSD, MIN MAX Q1 Q3 BOX, LNR EXPR PWR P2 P3, P4 FCX FCY SX SY, and SHW XYLN LIN 1V 2V, wrapping back to the first. The sixth page's last three keys repeat the first page's, so the everyday summaries stay one press away from the far pages.

The screenshot's data is the five pairs (1,2), (2,4), (3,5), (4,4), and (5,5): press + once for a length of five, type the X values, press ALPHA, and type the Y values.

15-2 One-variable statistics

For a worked example, put the eight values 2, 4, 4, 4, 5, 5, 7, 9 in the X column (press + four times, then type them).

1V (F1, elsewhere OneVar) computes the one-variable summary of the X column and answers a result screen: MEAN reads 5, MED reads 4.5, S SD reads 2.1380899352994, and P SD reads 2. S SD is the sample standard deviation, whose squared deviations divide by n-1; P SD is the population standard deviation, dividing by n. The median is the middle value, or the mean of the middle two when the count is even. EXIT leaves a result screen for the home screen, and STAT reopens the editor with the data kept.

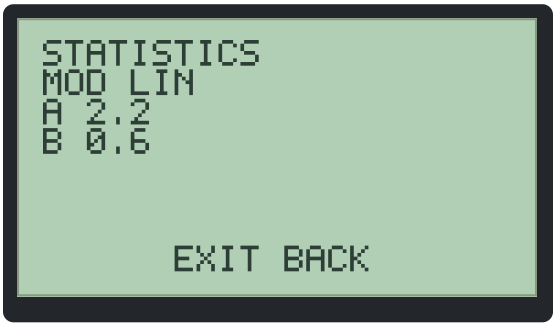
The second page's keys answer one figure at a time on the same kind of screen: MEAN and MED repeat the summary lines, VAR answers the sample variance 4.5714285714286 (the square of S SD; square P SD for the population variance), and SSD and PSD repeat the two standard deviations. The third page's MIN, MAX, Q1, and Q3 answer 2, 9, 4, and 6. The quartiles are the medians of the lower and upper halves of the sorted data, and an odd count leaves the middle value out of both halves: the column 1, 2, 3, 4, 5 answers 1.5 for Q1 and 4.5 for Q3.

Two-variable statistics and linear regression

15-3

Enter the five pairs from the editor screenshot above. 2V (F2, elsewhere TwoVar) computes the paired summary: MEANX reads 3, MEANY reads 4, and R, the correlation coefficient, reads 0.7745966692415.

LIN (F3, elsewhere LinR) fits the least-squares line through the pairs and answers a result screen: MOD LIN names the model, A, the intercept, reads 2.2, and B, the slope, reads 0.6, so the fitted line is y = 2.2 + 0.6x. The footer reads EXIT BACK, and the two ways off the screen differ: EXIT leaves for the home screen, and CLEAR returns to the editor. The screen stops at the coefficients: the correlation lives on the 2V screen alone, so read R there before or after the fit.



The linear regression of the five pairs

Watch degenerate data: a constant X column has no defined slope, and rather than an error the result screen answers A 0 and B 0, so treat an all-zero fit with suspicion and check the data.

15-4 The further regression families

The fourth soft-key page holds LNR, EXPR, and PWR and the polynomial fits P2 and P3, with P4 opening the fifth page. Each key fits its model to the pairs and answers the same result screen: MOD names the model, and the coefficients follow in the model's own terms.

LNR (F1, elsewhere LnR) fits the logarithmic model y = A + B ln x. For the four pairs (1,1), (2,3), (4,5), (8,7), whose y climbs by 2 per doubling of x, the screen answers MOD LN with A reading 0.9999999999992 and B reading 2.8853900817765, the machine's account of the exact fit A = 1, B = 2/ln 2.

EXPR (F2, elsewhere ExpR) fits the exponential model y = A e^(Bx). The pairs (0,3), (1,6), (2,12), (3,24) double per step from 3, and the screen answers MOD EXP with A reading 3.0000000000026 and B reading 0.69314718056, which is ln 2.

PWR (F3, elsewhere PwrR) fits the power model y = A x^B. The pairs (1,3), (2,12), (3,27), (4,48) lie on y = 3x^2, and the screen answers MOD POWER with A reading 3.000000000001 and B reading 1.9999999999983.

P2 and P3 (F4 and F5, elsewhere P2Reg and P3Reg) and the fifth page's P4 (F1, elsewhere P4Reg) fit least-squares polynomials of degree two through four. The coefficients come in ascending powers, A the constant upward: the pairs (0,1), (1,6), (2,17), (3,34) under P2 answer MOD P2 with A 1, B 2, and C 3, the parabola y = 1 + 2x + 3x^2 exactly, and the five pairs of y = x^4 at x = -2 through 2 under P4 answer MOD P4 with A 0 through D 0 and E 1.

Two guards protect the transformed fits. The logarithm asks for positive data, so LNR with a zero or negative X entry, EXPR with one in Y, or PWR with either answers the POSITIVE DATA NEEDE notice, its final letter fallen off the screen, short for positive data needed. A polynomial needs at least one pair per coefficient, three for P2 up to five for P4; fewer answers the NEED TWO SAMPLES notice, whose wording stays the same however many samples the model really wanted. Both notices show the usual CLEAR OR EXIT footer, but here either key dismisses to the home screen rather than back to the data, so press STAT to return to the editor.

Forecasting

15-5

Once a model is fitted, the fifth page's FCY (F3, elsewhere fcsty) forecasts y from x. Its input is the entry the editor is standing on: FCY reads the selected entry's X value, runs it through the model, and answers a FORECAST screen whose direction line reads X–>Y, the forecast above and the x it used below. With the five pairs fitted by LIN, stepping to INDEX 3 and pressing MORE four times then F3 answers 4 over 3, the fitted line's value at x = 3. To forecast at an x the data does not contain, fit first, then grow the columns by one and store the x you want; the fit is not recomputed until you press a regression key again.

FCX (F2, elsewhere fcstx) inverts the model: it reads the selected entry's Y value and answers the x that produces it, the direction line reading Y–>X. On the five pairs' linear fit, FCX at INDEX 1 (whose Y is 2) answers –0.33333333333333, the x where 2.2 + 0.6x = 2. For the two-coefficient models the inverse is solved directly; for P2 through P4 the machine searches the span between the data's smallest and largest X and answers the first crossing it finds in ascending order. When no x in that span produces the target y, the search gives up at the FCSTX NEEDS 2–COEFF notice, which dismisses to the home screen like the fitting guards above; an inverse forecast outside the data's range needs the two-coefficient families.

The FORECAST screen leaves like the result screens, EXIT for the home screen and CLEAR for the editor, and the forecast keeps either way: SHW below repaints it on demand.

Sorting and recalling

15-6

SX and SY (the fifth page's F4 and F5, elsewhere Sortx and Sorty) sort the pairs in place, ascending by the named column, and carry the other column along so the pairs stay intact. With X holding 3, 1, 4, 2 and Y holding 30, 10, 40, 20, SX leaves the columns reading 1, 2, 3, 4 and 10, 20, 30, 40: entry by entry, each y still rides with its x. The editor stays where it was, so step through the entries to see the new order.

SHW (the sixth page's F1, elsewhere ShwSt) repaints the last result screen, whether that was a summary, a regression, or a forecast, and is the way back to figures you left with EXIT. Fit the five pairs with LIN, leave for the home screen,

return with **STAT**, and SHW answers MOD LIN with A 2.2 and B 0.6 again. Before anything has been computed the key does nothing.

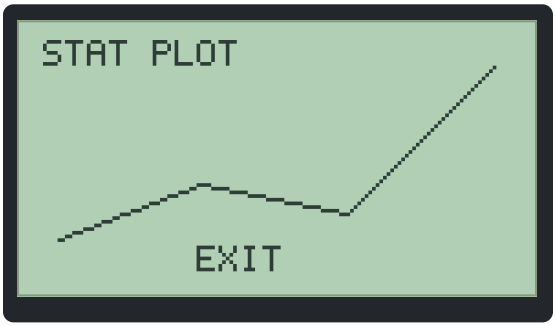
15-7 Statistical plots

Four soft keys turn the columns into pictures. Each plot draws under a STAT PLOT banner, scales itself so the data's smallest and largest values touch the edges of the plotting area (the graph window of chapter 4 plays no part), draws no axes, and leaves for the home screen with **EXIT**, as its footer says.

SCAT (**F4**, elsewhere Scatter) draws one dot per pair, X across and Y up. With the five pairs entered, the dots climb from the lower left to the upper right, with the dip at (4,4) visible on the way:

The scatter plot of the five pairs

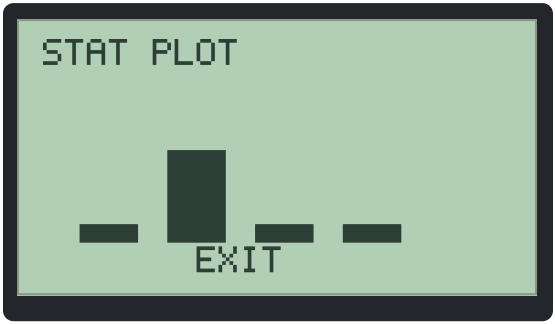
XYLN (the sixth page's **F2**, elsewhere xyline) draws the same axes-free frame but joins consecutive pairs with line segments in entry order, clipped to the frame. The pairs (1,2), (2,4), (3,3), (4,8) draw a rise, a dip, and a steep climb:



The connected line plot of four pairs

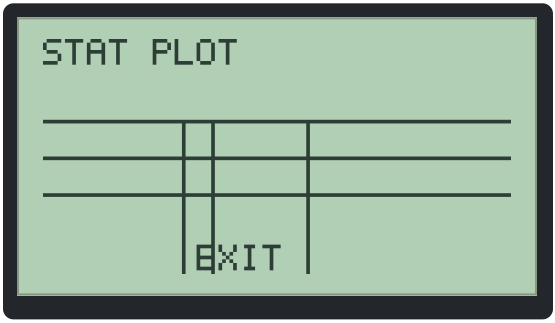
Entry order is drawing order, so a column that is not sorted by `X` draws a zig-zag; `SX` above puts the pairs in plotting order first.

`HIST` (F5, elsewhere `Hist`) draws a histogram of the `X` column alone, ignoring `Y`. It sorts the values into four equal-width bins spanning the range from minimum to maximum and draws a bar per bin, heights in proportion to the counts. For the eight-value column of the one-variable example the bins hold 1, 5, 1, and 1 values, so the second bar towers over the other three:



The histogram of the eight-value column

`BOX` (the third page's F5) draws the `X` column's quartile summary, scaled so the left and right edges stand for the minimum and the maximum: vertical bars mark the lower quartile, the median, and the upper quartile. In this release the drawing overgrows the classic box shape, the box's three horizontal lines running the full width of the screen and the three vertical bars dropping from the top line to the bottom edge, so read the bars' left-to-right positions and let the shape go. For the eight-value column the bars sit at 4, 4.5, and 6 between edges standing for 2 and 9:



The box plot of the eight-value column

A plot with nothing to draw draws nothing: a single-entry or constant `X` column answers an empty `STAT PLOT` frame rather than an error.

CHAPTER

16

Calculator Programming

*The **PRGM** key leads to a small but complete programming environment: four stored programs, a line-at-a-time editor, and a runner that shares the expression engine of the home screen. This chapter walks the list, the editor, and every instruction the language understands. Every program in it was run in the emulator and its output quoted exactly as the screen shows it, and where the editor cannot yet type an instruction the runner accepts, the text says so plainly.*

The program list

16-1

Press [PRGM] to open the program list; the PGM soft item on the home screen's second menu page ([MORE] [F5], chapter 1) leads to the same place:

The program list on a fresh machine

Under the PROGRAMS banner sit the four program slots, each reading <EMPTY> on a fresh machine, with the > cursor marking the selected slot. [▲] and [▼] move the cursor, wrapping at both ends, and [EXIT] returns to the home screen. Four slots is the whole store: Free85 holds at most four programs, and this list is all of them.

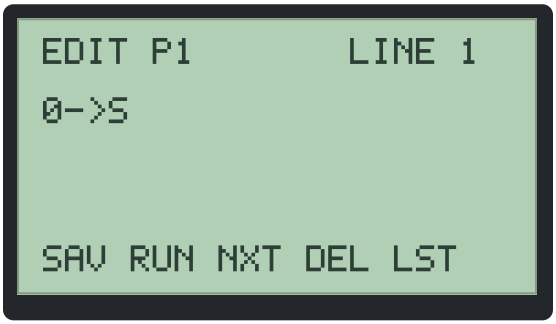
The soft keys NEW EDT RUN REN DEL do the work:

- F1 **NEW** creates a program in the selected slot, named `P1` through `P4` after the slot, and opens it in the editor. The two keys converge on a filled slot: there `NEW` simply opens the program, the same as `EDT`.
- F2 **EDT** opens the selected program in the editor, creating it first if the slot is empty.
- F3 **RUN** runs the selected program. On an empty slot it answers the full-screen `NO PROGRAM` notice, with the usual `CLEAR OR EXIT` way back (chapter 1).
- F4 **REN** renames the selected program: the `RENAME PROGRAM` screen loads the current name into an entry line, and its footer `ENTER SAVE` says how to finish. Names hold up to seven characters, typed like any program text (letters with ALPHA); an empty or overlong name answers the `BAD NAME` notice, and EXIT abandons the rename. A renamed slot shows its new name in the list.
- F5 **DEL** deletes the selected program on the spot, with no confirmation, exactly like the bulk `PGM` clear in Chapter 18: Memory Management.

Programs persist: they survive leaving the screen, and a power cycle brings all four slots back exactly as you left them.

16-2 The program editor

`NEW` on the first slot opens the editor. The capture below shows it after typing the first line of this chapter's worked example:



The program editor holding a first line

The banner names the program and the line: `EDIT P1` and `LINE 1`. A program is eight lines of up to 48 characters each, and the editor shows one line at a time; the line number in the banner is your position.

Typing works like the home entry line (chapter 1), with the same insertions: SIN inserts `SIN(`, STO▶ inserts `->`, x-VAR inserts `X`, and letters are typed with ALPHA, so a bracketed letter such as D means ALPHA then the key carrying that letter. DEL deletes, 2nd DEL toggles insert and overwrite, CLEAR empties the line, and ◀ and ▶ move the cursor. One difference is worth flagging: in this editor 2nd 0 types a space character, where on the home screen the same keys open the character palette.

The editor has no insert menus. MORE answers nothing here, and neither the catalog (2nd CUSTOM) nor the `TEST` menu of chapter 3 (2nd 2) opens, so every keyword is spelled out letter by letter with ALPHA. That closed door leaves three characters short: `=`, `<`, and `>` live only in menus and in the character palette, and the palette is the very thing 2nd 0 does not open here. A `>` can still be manufactured, because STO▶ inserts the two characters `->` and ◀ DEL ▶ removes the `-`, leaving a bare `>` behind. No trick reaches `=` or `<`, so although the runner understands comparison conditions such as `REPEAT A=3` and the skip instruction `DS< V,e`, this release's editor cannot key them in; until a firmware release opens a way to type those characters, build conditions from arithmetic, as the examples below do.

Moving between lines always saves the line you are leaving:

- ENTER, ▼, or F3 **(NXT)** move down one line, stopping at line 8.
- ▲ moves up one line, stopping at line 1.
- F1 **(SAV)** saves the line and stays put.
- F4 **(DEL)** deletes the whole current line and pulls the lines below it up one place.
- F5 **(LST)** or EXIT save and return to the program list.
- F2 **(RUN)** saves and runs the program immediately.

The language at a glance

16-3

A program is one statement per line. A statement is either an instruction from the table below or a bare expression, and expressions run through the same engine as the home screen, so everything in Chapter 3 (Mathematics, Calculus, and Comparisons) works here: `2+3->A` stores and `SIN(0)` evaluates. Conditions are numeric, nonzero meaning true; the comparison operators exist in the engine but

their characters mostly cannot be typed here, as the editor section explains. An instruction keyword is followed by one space, then its arguments.

Instruction	Meaning
`DISP e`	evaluate `e` and show it on the run screen
`INPUT V`	ask for a number and store it in variable `V`
`PROMPT V`	the same, with the variable named in the banner
`INPST S`	ask for text and store it in string register `S`
`IF e` … `ELSE` … `END`	run a block when `e` is nonzero
`WHILE e` … `END`	repeat a block while `e` is nonzero
`REPEAT e` … `END`	repeat a block until `e` is nonzero
`FOR V,a,b[,s]` … `END`	count `V` from `a` to `b`, stepping `s`
`LBL name` / `GOTO name`	mark a line and jump to it
`IS> V,e` / `DS< V,e`	step `V`, skipping a line on a comparison
`MENU one,two,...`	suspend on a soft-key chooser of labels
`GETKEY V`	store the last key's code in `V`, without waiting
`PAUSE`	wait for any key
`OUTPT r,c,text`	print `text` at row `r`, column `c`
`CLLCD`	clear the display
`CALL n`	run program `n` (1 through 4), then come back
`RETURN`	leave the current program at once
`STOP`	end the run
`GRAPH e`	store `e` as the active equation and end the run on its plot
`DISPG`	end the run on the graph screen
`STTOEQ S,n` / `EQTOST n,S`	copy a string to equation slot `n`, and back
`LSET i,e` / `LGET i,V`	write and read entry `i` of list `A`
`MSET r,c,e` / `MGET r,c,V`	write and read cell `r,c` of matrix `A`
`VSET i,e` / `VGET i,V`	write and read entry `i` of vector `A`

Instruction	Meaning
`VOUT text` / `VIN V`	write to and read from the virtual device
`PRTSCRN`	print the display to the virtual device
`CAT command`	run a catalog function (`CAT` is optional)
`COLL n` / `STATC n`	end the run on a collection or statistics result
`SOLVER e` / `GMODE n`	end the run on the solver or the graph mode `n`

If you are arriving from another calculator's manual, the spellings map directly: `DISP` covers `Disp`, `INPUT` covers the numeric form of `Input` (appendix A catalogues it as `Input-number`), and `WHILE`, `FOR`, `ELSE`, `END`, `RETURN`, and `STOP` cover `While`, `For`, `Else`, `End`, `Return`, and `Stop`. The 2.10 additions map the same way: `LBL`, `GOTO`, `REPEAT`, and `MENU` cover `Lbl`, `Goto`, `Repeat`, and `Menu`; `GETKEY`, `OUTPT`, `PAUSE`, `PROMPT`, `INPST`, `CLLCD`, `DISPG`, and `PRTSCRN` cover `getKy`, `Outpt`, `Pause`, `Prompt`, `InpSt`, `ClLCD`, `DispG`, and `PrtScrn` (appendix A catalogues the text form of `Input` as `Input-string`); `IS>` and `DS<` keep their spellings. The one structural difference is the conditional: an `IF` line opens its block directly, with no separate `Then` line, so a three-line `If/Then/End` block elsewhere is a two-line `IF/END` block here.

The table's last rows reach the rest of the machine, and the closing sections walk them; appendix A files that reach as `all-math-from-programs`, `all-graph-from-programs`, `all-collection-from-programs`, and `all-statistics-from-programs`.

A first program

16-4

The worked example sums the numbers 1 through 5. Press PRGM F1 to create `P1`, then type these six lines, pressing ENTER after each to move on (spaces are 2nd 0):

Press F2 (RUN) and the run screen takes over:

The run screen after the summing program finishes

Reading from the top: `RUN P1` names the program, `LINE 6` is the line the runner reached, the output line shows `15`, the sum of 1 through 5, the status reads `DONE`, and the footer `ON STOP` names the panic button. The output line shows the most recent `DISP` only, so a program that displays many values leaves the last one on screen.

From the run screen, PRGM returns to the program list, and EXIT from the list goes home. Pressing EXIT, CLEAR, or ON on the run screen instead marks the run stopped, as the stopping section below describes.

16-5 Conditions

`IF` evaluates its expression and runs the following block when the value is nonzero; a zero value falls through to the `ELSE` block, if there is one, and `END` closes the conditional. This program:

```
0->A
IF A
DISP 1
ELSE
DISP 2
END
STOP
```

answers `2` on the run screen: `A` is zero, so the `ELSE` branch runs. Change the first line to `1->A` and the same program answers `1`. Since the editor cannot type `=` or `<`, phrase tests arithmetically: `A-5` is nonzero exactly while `A` differs from 5.

Loops

16-6

`WHILE` re-tests its expression before every pass and leaves the block when the value is zero. A countdown:

```
3→A
WHILE A
A−1→A
END
DISP A
STOP
```

answers `0`, the value of `A` when the test finally failed.

`REPEAT` (elsewhere `Repeat`) is its mirror: the block repeats until the expression is nonzero. One real difference from its namesake: elsewhere `Repeat` tests after the body, so the body always runs at least once, while here the test sits at the `REPEAT` line and runs before every pass, the first included. This program:

```
0→A
REPEAT A
A+1→A
END
DISP A
STOP
```

answers `1`: the body ran once, `A` became nonzero, and the loop ended. Change the first line to `3→A` and it answers `3`, the body never entered, because the condition was already satisfied at the first test.

`FOR` is the counted loop. The variable is any letter; the start, the end, and an optional step are evaluated expressions rather than literal digits, and each must come out a whole number within the signed 16-bit range. The step defaults to 1 and may be negative, in which case the count descends; a range already empty in the chosen direction runs its block no times at all. `FOR A,1,3` runs its block with `A` at 1, 2, and 3, and leaves `A` at 3 afterwards. This program:

```
FOR A,1,3
DISP A*A
END
STOP
```

displays the squares 1, 4, 9 in turn and finishes with `9` on the output line, the last `DISP` standing. Because the bounds are expressions, `FOR A,1+1,NCR(5,2)+2,2` counts 2, 4, 6, 8, 10, 12, and loops nest as you would expect: `FOR A,1,2` wrapped around `FOR B,2,4,2` finishes with `A*10+B` reading `24`.

Three things are refused rather than guessed at, each with `DOMAIN ERROR`: a step of zero, which would never finish; a bound that is not a whole number, as in `FOR A,1.5,3`; and a bound outside the signed 16-bit range, as in `FOR A,1,40000`.

16-7 Labels, jumps, and skips

`LBL name` (elsewhere `Lbl`) marks a line and does nothing when the runner walks over it; `GOTO name` (elsewhere `Goto`) jumps to the matching `LBL` anywhere in the program. Names run up to 16 characters. This program:

```
GOTO SKIP
DISP 1
LBL SKIP
DISP 7
STOP
```

answers `7`, the `DISP 1` jumped over. A `GOTO` whose name has no `LBL` stops the run at the error notice naming its own line: a program whose second line is `GOTO MISSING` stops at `ERROR LINE 2`.

`IS> V,e` adds 1 to variable `V`, then skips the next line when the new value exceeds `e`; `DS< V,e` subtracts 1 and skips when the new value has dropped below `e`. Both pair naturally with `GOTO` on the line they guard. This program:

```
0→A
IS> A,0
DISP 0
DISP A
STOP
```

answers `1`: the increment took `A` to 1, 1 exceeds 0, and the `DISP 0` was skipped; the `IS>` line itself is typeable through the **STO▶** trick from the editor section. The runner carries the mirror `DS<` too, though nothing you can type on this release reaches it, because no trick produces the `<`; for the record, its behaviour:

```
2->A
DS< A,2
DISP 0
DISP A
STOP
```

answers `1` from `DISP A`, the decrement landing below 2. Until a firmware release opens a way to type <, count downward with `WHILE` and a stored variable instead.

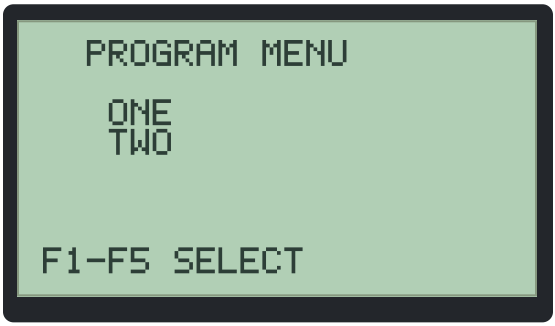
Program menus

16-8

`MENU` (elsewhere `Menu`) lists up to five names and suspends the run on a chooser. Each name must match a `LBL` somewhere in the program, and the soft key jumps there. With `P1` holding:

```
MENU ONE,TWO
LBL ONE
DISP 1
STOP
LBL TWO
DISP 2
STOP
```

running suspends on the chooser: the banner reads `PROGRAM MENU`, the names are listed down the screen, and the footer reads `F1-F5 SELECT`:



The program menu chooser offering two labels

F2 picks `TWO`, the run jumps to `LBL TWO`, and the run screen answers `2`. Keys other than F1 through F5 redraw the chooser, except ON, EXIT, and CLEAR, which stop the run. The bounds are enforced at run time: six names or an empty list

stop the run at the ERROR LINE notice on the MENU line, and picking a name with no matching LBL stops it like an unresolved GOTO. Names follow the label rule, 16 characters at most.

16-9 Asking for a number

INPUT names one variable, A through Z. When the runner reaches it, the run pauses on a dedicated screen: the banner names the variable, INPUT A, an empty entry line waits, and the footer reads ENTER VALUE. Type a number, using the digits, [.], and [(-)] as on the home screen, and press [ENTER]; the value is stored and the run carries on. This program:

```
INPUT A
DISP A*2
STOP
```

pauses at INPUT A; typing [6] [ENTER] resumes the run, and the output line answers 12. An entry that does not parse as a number (a bare [.], say) stops the run at the ERROR LINE 1 notice, naming the INPUT line, and [EXIT] or [ON] on the input screen abandons the run the same way as stopping it.

PROMPT V (elsewhere Prompt) asks the same way on the same screen; the difference is the banner, which names the keyword instead of the word INPUT: PROMPT A for variable A. Swap the first line above for PROMPT A and the same [6] [ENTER] answers 12.

16-10 Asking for text

INPST S (elsewhere InpSt) suspends the run on a text-entry screen and stores what you type into string register A or B of Chapter 9 (Strings and Characters); any other register name stops the run at the ERROR LINE notice. The screen's footer reads ENTER TEXT, and its banner names the register after the keyword: INPUT STRING A for register A. Letters are typed with [ALPHA], digits and operators directly, [x-VAR] inserts X, and [ENTER] stores; the register's 31-character ceiling from chapter 9 applies. The stored text lands in the strings editor exactly as typed, and the equation bridge below turns it into something the calculator can run.

Reading the keyboard

16-11

`GETKEY V` (elsewhere `getKy`) does not wait: it stores the code of the most recently pressed key into `V` and moves on, storing `0` when nothing has been pressed since the run began. Codes 1 through 50 number the physical keys by position, **F1** at `1` across to **ENTER** at `50`; **SIN**, for example, is `22`. The canonical use is a polling loop, and this one is typeable as written:

```
REPEAT A
GETKEY A
END
DISP A
STOP
```

The loop spins while `A` is zero, so the run waits until you press something; pressing **SIN** answers `22` on the run screen.

Writing the screen

16-12

`CLLCD` (elsewhere `ClLCD`) clears the display to blank. `OUTPT r,c,text` (elsewhere `Outpt`) prints text at row `r` (0 through 7) and column `c` (0 through 20). Its third argument is taken literally, not evaluated: `OUTPT 1,0,2+2` prints `2+2`, not `4`. Text runs up to 23 characters and clips at the right edge rather than wrapping, one positioned string shows at a time (a second `OUTPT` replaces the first), and a row, column, or text outside the bounds stops the run at the `ERROR LINE` notice. The program `OUTPT 3,12,HELLO` ends with the word alone on the cleared screen:

Positioned output from OUTPT

PAUSE (elsewhere Pause), with no argument, suspends the run on a screen reading PAUSED over PRESS A KEY; any key resumes, except [ON], [EXIT], and [CLEAR], which stop the run. It is the natural partner of OUTPT and CLLCD, holding a composed screen still long enough to read.

16-13 Programs calling programs

CALL runs another of the four programs by its slot number and comes back to the next line when that program ends or reaches RETURN. Variables are shared, so a called program hands results back by storing them. With P1 holding:

```
CALL 2
DISP A
STOP
```

and P2 holding:

```
7->A
RETURN
```

running P1 answers 7: the call ran P2, which stored 7 in A and returned. CALL on an empty slot stops the run with an error, and RETURN in the top-level program simply ends the run. Calls nest up to four deep, as the limits section below records.

16-14 Stopping, and when it goes wrong

STOP ends the run and leaves the status at DONE. For a program that will not end on its own, the footer's promise holds: [ON] stops the run at once. Key in the two-line program WHILE 1 and END, run it, and the status shows RUNNING with the line number ticking; press [ON] and the screen answers the stopped notice naming whichever line the runner was on, STOPPED LINE1 or STOPPED LINE2 for this two-line loop. [EXIT] and [CLEAR] stop a run the same way, so the deliberate exits from a finished run screen are [PRGM] to the list and [EXIT] from there. The waiting instructions are covered too: [ON] interrupts INPUT, PROMPT, INPST, PAUSE, and a MENU chooser on the spot, and a stopped run never alters the program source.

A line the runner cannot make sense of stops the run with an error notice naming the line: a program whose second line is the stray text HELLO runs its first line,

then stops at `ERROR LINE 2`. Fix the line in the editor and run again; the source is never altered by a failed run.

Strings and equations 16-15

`STTOEQ S,n` copies string register `S` (`A` or `B`) into graph equation slot `n` (1 through 3) of Chapter 4 (Cartesian Graphing, Drawing, Formats, and Persistence) and switches that slot on for plotting; `EQTOST n,S` copies a slot back into a register. Chapter 9 records the pair's ancestry (elsewhere `St->Eq` and `Eq->St`); this is their home, because they live in the program language only. The trip is exact both ways, and `INPST` above completes the circle by turning typed text into a plottable equation. This program:

```
INPST A
STTOEQ A,1
EQTOST 1,B
STOP
```

pauses for text; typing `X+2` and pressing **ENTER** ends the run with register `A` holding `X+2`, equation slot 1 holding the same text and enabled, ready for **GRAPH** to plot the line, and register `B` holding the copy that `EQTOST` read back. A slot outside 1 through 3 or a register outside `A` and `B` stops the run at the `ERROR LINE` notice, and the 31-character register ceiling bounds the equation text.

The virtual device 16-16

Three instructions talk to the built-in virtual device, a 25-byte buffer that stands in for a physical device port and is always ready to use. `VOUT text` writes its literal text, up to 25 bytes, into the device buffer; `VIN V` reads the buffer back, parses it as a number, and stores it in `V`; `PRTSCRN` (elsewhere `PrtScrn`) prints the display to the device, which in this release records the eight characters `LCD:1024`, the display's size in bytes standing in for pixel data. This program:

```
VOUT 42
VIN A
DISP A
PRTSCRN
STOP
```

answers 42 on the run screen, the number having made the round trip through the device, and leaves LCD:1024 in the buffer. VOUT 3.5 followed by VIN A stores 3.5, and text longer than 25 bytes stops the run at the ERROR LINE notice.

> **HARDWARE**
> VOUT, VIN, and PRTSCRN drive the built-in virtual device standing in for the physical port that CBL-style external forms of Input and Outpt expect; physical hardware validation is reported separately.

16-17 Reaching the rest of the calculator

Everything the expression engine offers is available inside program expressions, and that now includes the whole command catalog of appendix A: SQRT(9)->A stores 3 from a program line exactly as it does on the home screen. The CAT prefix marks a catalog call explicitly but changes nothing: CAT SQRT(9)->A and SQRT(9)->A are the same statement.

The collection bridges move single values without leaving the run:

- **LSET i,e and LGET i,V** write and read entry i (1 through 8) of list A, the list the editor of Chapter 12 (Lists) shows, growing the list when you write past its length. The program LSET 2,42, LGET 2,B, DISP B answers 42, and afterwards the list editor shows 42 at INDEX 2.
- **MSET r,c,e and MGET r,c,V** do the same for matrix A of Chapter 13 (Matrices and Vectors), rows and columns 1 through 3.
- **VSET i,e and VGET i,V** do the same for vector A, entries 1 through 3: VSET 2,7, VGET 2,A, DISP A answers 7.

The screen-based tools are reached by commands that end the run, the same one-way door in every case: the program cannot take the results back, but the screen you land on holds them.

- **GRAPH e** stores e as equation slot 1, enables it, and ends the run on the graph screen with the plot drawn: GRAPH X leaves the machine on the plotted line, slot 1 holding X. Like DISPG it does not return, so lines after it never run.
- **DISPG** (elsewhere DispG) also ends the run on the graph screen, drawing whatever equations are already stored and enabled; the difference is only that GRAPH stores its expression first while DISPG plots the current set. Neither

returns: lines after either never run, so both are closing statements, not display steps.

- **COLL n** runs collection operation n (0 through 17) and ends the run on that editor's result: codes 0 through 4 are the list keys (dimension, fill, descending sort, and the two vector conversions), 5 through 7 the vector keys (dimension, fill, magnitude), and 8 through 17 the matrix keys (echelon reduction, the three norms, condition number, LU factors, eigenvalues, eigenvectors, dimension, fill). LSET 2,42 then COLL 2 lands on the list screen with the sorted result R showing 42 at INDEX 1, the descending sort having carried the written value ahead of the zeros; the result's SIZE 4 is a reminder that a fresh machine's list already holds four entries. The complex editor of chapter 11 is not among the codes.
- **STATC n** runs statistics operation n (0 through 10) over the columns of Chapter 15 (Statistics and Statistical Plots), which are lists A and B: 0 and 1 are 1V and 2V, 2 through 8 the regression fits LIN, LNR, EXPR, PWR, P2, P3, P4, and 9 and 10 the sorts SX and SY. With the chapter's five pairs entered, STATC 2 ends the run on the statistics screen reading MOD LIN with A 2.2 and B 0.6.
- **SOLVER e** loads e into the solver workspace of Chapter 14 (Solving Equations) and ends the run there: SOLVER X−2 lands on the SOLVER screen with F= X−2 and VAR X ready to solve.
- **GMODE n** switches the graph mode, 0 through 3 for function, polar, parametric, and differential-equation, and ends the run on the graph screen in that mode.

An opcode outside its range stops the run at the ERROR LINE notice before any screen changes.

Limits

16-18

The environment's bounds are fixed in this release, and the editor and runner enforce all of them:

- four programs, of eight lines of 48 characters each;
- control frames (IF, WHILE, FOR, and REPEAT blocks) nest eight deep;
- calls nest four deep;
- label and menu names run up to 16 characters, and a menu offers at most five of them;

- `OUTPT` text runs up to 23 characters, on rows 0 through 7 and columns 0 through 20;
- string registers are `A` and `B`, equation slots 1 through 3, and the device buffer holds 25 bytes.

Exceeding a nesting bound stops the run with the `ERROR LINE` notice on the line that went too deep, and every waiting instruction remains interruptible by **ON** whatever the depth.

CHAPTER

17

Worked Application Examples

The reference chapters cover one feature at a time; this chapter puts them together. Four complete examples follow, one from graphing, two from statistics, and one from programming, each solved end to end with every keystroke listed. All four were run start to finish in the emulator on a fresh machine, and every quoted figure is what the screen showed.

Example 1: finding and checking a root

17-1

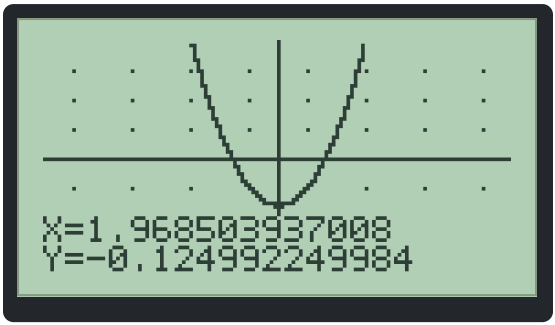
The task: study the parabola y = x^2-4, locate a root, and check the answer numerically.

Plot the function. The home entry line is the equation editor (Chapter 4: Cartesian Graphing, Drawing, Formats, and Persistence), so type [x-VAR] [x²] [-] [4] and press [GRAPH]. The expression X^2−4 is stored as Y1 and the parabola draws, column by column. Let the plot run to completion so the whole curve is on screen for the trace below.

Frame the view. The zoom keys replot immediately: press [+] to zoom in and watch the window halve to -5 to 5, then press [2nd] [+], the standard-zoom key, to restore the -10 to 10 window this example uses.

Trace towards the root. The [▶] key traces along the curve from the centre column, one column per press, with the exact coordinates in the readout at the

bottom. Press ▶ twelve times and the readout shows the curve approaching its positive root near x = 2:



Tracing X^2-4 near the root at x=2

The readout gives `X=1.968503937008` and `Y=-0.124992249984`: just short of x = 2, the curve sits a fraction below the axis.

Publish a root. Press F1, the root key, and the calculator answers on the home screen with `= -2` and the residual line `R=0`. The search scans the window from its left edge, so it lands on the leftmost root, x = -2; by the parabola's symmetry the traced root is its mirror image at x = 2.

Check both numerically. From the home screen (press CLEAR to empty the entry line), the calculus commands of Chapter 3: Mathematics, Calculus, and Comparisons read the same stored equation. Type `EVAL(2)` with the letters (E V A L, chapter 1's ALPHA convention) and press ENTER: the answer is `= 0`, confirming x = 2 is a root on the nose. Then `NDER(2)` answers `= 4`, the slope 2x at the root, so the curve crosses the axis there rising at slope 4.

One plot, a trace, one soft key, and two home-screen commands: the graph screen found the root and the calculus commands proved it.

17-2 Example 2: fitting and forecasting a line

The task: fit a straight line through a small paired dataset and forecast the next value.

Enter the data. The dataset is four pairs: (1,2), (2,3), (3,5), and (4,6). Press STAT to open the statistics editor of Chapter 15: Statistics and Statistical Plots. A fresh machine already holds four entries, so no resizing is needed: type 1 ENTER

2 ENTER 3 ENTER 4 ENTER to fill the X column, press ALPHA to switch to the Y column, and type 2 ENTER 3 ENTER 5 ENTER 6 ENTER.

Fit the line. Press F3 (LIN) and the regression result screen answers:

The linear regression of the four pairs

Under the STATISTICS banner, MOD LIN names the model, A, the intercept, reads 0.5, and B, the slope, reads 1.4: the fitted line is y = 0.5 + 1.4x. The footer EXIT BACK names chapter 15's two ways off a result screen, EXIT to the home screen and CLEAR back to the editor.

Judge the fit. The result screen stops at the coefficients, so press CLEAR to return to the editor and F2 (2V): the paired summary answers R reading 0.9899494936611, a strong fit.

Forecast. Chapter 15's FCY key forecasts from an entry in the data columns, as example 4 shows; for a one-off x the two coefficients are quicker typed by hand: press EXIT to leave for the home screen, press CLEAR, and evaluate the model at x = 5 by typing .5+1.4*5 and pressing ENTER. The answer line reads = 7.5, the fitted forecast for the fifth pair. Four pairs, two soft keys, and one typed line: the editor held the data, LIN named the model, and the home screen turned it into a forecast.

Example 3: a factorial program

17-3

The task: compute 5! with a program, then check the answer with a built-in function.

Write the program. Press PRGM F1 to create P1 in the editor of Chapter 16: Calculator Programming, and type these six lines, pressing ENTER after each (letters with ALPHA; the space is 2nd 0):

Line	Text	Keys
1	`1->F`	[1] [STO▶] [F]
2	`FOR A,1,5`	[F] [O] [R] [2nd] [0] [A] [,] [1] [,] [5]
3	`F*A->F`	[F] [×] [A] [STO▶] [F]
4	`END`	[E] [N] [D]
5	`DISP F`	[D] [I] [S] [P] [2nd] [0] [F]
6	`STOP`	[S] [T] [O] [P]

Line 1 seeds the running product, the `FOR` loop multiplies it by 1 through 5, and `DISP` publishes the result.

Run it. Press [F2] (`RUN`):

The factorial program's run screen

The run screen answers `120` on its output line with the status `DONE`: 5! computed by four working lines.

Check it. Press [PRGM] to return to the list, [EXIT] to go home, and [CLEAR] to empty the entry line the run left behind. The factorial function of chapter 3 sits on the `MATH` menu, so press [F1] [F3] to insert `FACT(`, type [5] [)], and press [ENTER]: the answer is `= 120`, matching the program exactly. The program and the function agree because they share one arithmetic: the same fourteen-digit engine evaluates program lines and home entries alike.

17-4 Example 4: a curve through a projectile's flight

The task: a ball thrown straight up was measured once a second: at 1 s it stood at 25 m, at 2 s at 40 m, at 3 s at 45 m, and at 4 s at 40 m. Fit a quadratic to

height against time and forecast the height at 5 s.

Enter the data. Press [STAT] and fill the editor's four entries as in example 2: type [1] [ENTER] [2] [ENTER] [3] [ENTER] [4] [ENTER] into the X column for the seconds, press [ALPHA] to switch to the Y column, and type [2] [5] [ENTER] [4] [0] [ENTER] [4] [5] [ENTER] [4] [0] [ENTER] for the heights.

Fit the parabola. The polynomial fits sit on the editor's fourth soft-key page (chapter 15), so press [MORE] three times and press [F4] (P2). The result screen answers MOD P2 with A 0, B 30, and C −5, coefficients in ascending powers: the fitted curve is h = 30t - 5t^2, a launch speed of 30 m/s against a pull of 10 m/s^2, peaking at 45 m at t = 3 exactly as the data suggests.

Forecast the fifth second. Chapter 15's FCY key forecasts y from the x of the entry the editor is standing on, so give it an entry holding 5. Press [CLEAR] to return to the editor, [ALPHA] to switch back to the X column, and [+] to grow the columns to five entries; press [▼] four times to reach INDEX 5 and type [5] [ENTER] to store the time. [ENTER] wraps the selection back to INDEX 1, so press [▼] four times to stand on the new entry again. The soft keys kept their page while you edited, so a single [MORE] reaches the fifth page: press [F3] (FCY) and the FORECAST screen answers the direction line X–>Y with the forecast 25 above the 5 it used. The model puts the ball at 25 m as the fifth second ends, twenty below its peak and falling; [EXIT] leaves for the home screen. One editor, one polynomial key, and one forecast key: the forecast example 2 typed by hand, this example reads straight off a screen.

CHAPTER

18

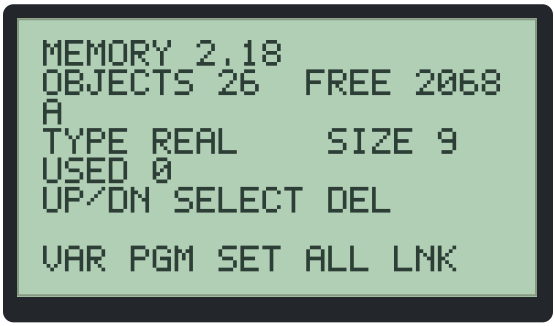
Memory Management

Chapter 2: Variables and Stored Data introduced the places Free85 stores your data; this chapter is about looking after them. The memory browser shows every stored object with its type named in words and its exact size in bytes, keeps a running account of the store's used and free space, deletes objects one at a time, performs the bulk clears and resets, and hands objects to the link screen of Chapter 19: Calculator Linking. Behind it sits the typed object store, in its 2.0 format, whose capacity rules and persistence guarantees close the chapter.

Opening the memory browser

18-1

The [+] key's shifted function is `MEM`. Press [2nd] [+] and the browser takes over the screen:



The memory browser

The same screen opens from the home screen's `MEM` soft key ([F4]) and from the `MEM` soft key on the system mode screen ([2nd] [MORE] [F5]), so it is never more than two presses away. [EXIT] returns you to the home screen. Appendix A catalogues this chapter's workflow as `memory-by-type`, `object-size`, `individual-delete`, `reset`, and `leave-memory-screen`.

Reading from the top: the title MEMORY 2.10 names the running release, and the second line is the store's account of itself. OBJECTS 26 counts every object it currently holds, and FREE reports the unused bytes of its data pool. On a fresh machine the free figure is 22,016, and here the 21-column screen shows its limits: the last digit falls off the right edge, so the line reads FREE 2201 until enough is stored to bring the figure under five digits.

The three lines beneath describe the selected object: its name A, then TYPE REAL beside SIZE 9, its kind in words and its exact size in bytes, and then USED 0, the bytes of the data pool the store's objects occupy in total. On a fresh machine those twenty-six objects are the reserved variables A through Z, which live in their own permanent places rather than the pool, so USED starts at 0. The hint UP/DN SELECT DEL and the soft labels VAR PGM SET ALL LNK list everything the screen can do.

18-2 Browsing by type and size

▲ and ▼ step the selection through the directory, stopping at both ends rather than wrapping; press ▼ once and the middle lines read B with TYPE REAL and SIZE 9. Every entry shows the same facts, so the browser doubles as a memory-usage display: the sizes are the store's own accounting, byte for byte, and a real number is nine bytes.

The browser names every kind of object the store can hold with a word: REAL, COMPLEX, LIST, MATRIX, VECTOR, STRING, EQUATION, PROGRAM, CONSTANT, GRAPH DB, and PICTURE cover the eleven kinds of chapter 2, several of which, PROGRAM among them, stay reserved for data that today lives elsewhere. An entry the firmware cannot place is listed as UNKNOWN, and if the store were ever empty the browser would say NO OBJECTS.

Beyond the reserved reals, the directory fills through the calculator's own workflows: Chapter 8 (Physical and User Constants and Conversions) stores user constants, and chapter 4's graph screen stores pictures and graph databases. To see the accounting move, plot any equation, store the image with chapter 4's StPic (CUSTOM, MORE MORE, F3), and reopen the browser: the count reads OBJECTS 27 and the total climbs to USED 1024. Step down to the new entry and it reads PIC1 with TYPE PICTURE; its size, 1,024 bytes, is one digit more than the SIZE column can show, so the line reads SIZE 102 with the final digit off the screen edge, and the USED line beneath carries the full figure.

Deleting one object

18-3

DEL deletes the selected object. For the reserved variables A through Z deletion clears the value back to 0 and keeps the directory entry, so the object count stays at 26 and the letter remains usable. Try it: store 5–>A, open the browser with 2nd +, press DEL, then EXIT and evaluate A. The answer is = 0.

For an ordinary object, deletion removes the directory entry and returns its bytes to the free pool immediately: DEL on the stored PIC1 above drops the count back to OBJECTS 26, returns the total to USED 0, and moves the selection to the previous entry.

Bulk clears and resets

18-4

The five soft keys act on whole categories at once. None of them asks for confirmation, so read this list before experimenting:

- F1 **VAR** clears all of A through Z to 0 in one press and confirms with a full-screen VARIABLES CLEARED notice; CLEAR or EXIT then returns to the home screen. The five numeric memories M1 through M5 are not touched.
- F2 **PGM** empties all four program slots and confirms with PROGRAMS CLEARED. Chapter 16: Calculator Programming covers what lives there; its programs keep to their four slots rather than the directory, never appearing as TYPE PROGRAM entries, and this clear is the browser's whole hold on them.
- F3 **SET** resets the system settings: the angle mode returns to RAD, the display format to AUTO, and the contrast to its default. The screen stays on the browser, and stored data, variables, and memories all survive.
- F4 **ALL** is the full reset. The calculator restarts on the spot, exactly as at first boot: variables, numeric memories, programs, and settings are all gone, and a fresh object store is built. There is no confirmation step, so treat F4 with respect.
- F5 **LNK** opens the link screen of chapter 19 standing on the same object the browser had selected, ready to mark it for transfer.

HARDWARE

the browser, clears, and resets all run in the emulator, and the LNK screen leads to the emulator-run link workflow of chapter 19; physical hardware validation is reported separately.

18-5 Capacity and accounting

The object store keeps a directory of up to sixty-four entries backed by a compacting data pool of 22,016 bytes. The accounting is exact by design:

- deleting an object closes the gap in the pool at once and slides the later objects down, so free space is always one contiguous run;
- resizing an object moves everything after it and commits only after the capacity check succeeds;
- a request that cannot fit, whether the directory is full or the pool is short, is refused outright and the store is left exactly as it was.

The practical consequence is the best kind of boring: there is no fragmentation to manage, the SIZE, USED, and FREE figures in the browser add up to the truth, and running out of memory produces a refusal rather than a corrupted store. Free85 never silently overwrites your data to make room.

18-6 Persistence and migration

The store is built to survive. Its header is validated on every start, and a valid store comes through a warm restart byte for byte, so switching the calculator off with 2nd ON and on again costs you nothing that was stored.

Upgrading from a Free85 1.0 state is handled the same way. The store format carries a version number (currently 13), so the firmware recognises an older state on sight. It then keeps all existing data in place, rebuilds the typed directory around it, and advances the version number only as the final step. If a reset or power loss interrupts the process, the unchanged version number simply causes the migration to run again from the start; it cannot half-complete. Should the store's header itself ever be found corrupt, the directory is rebuilt and the values of A through Z are preserved.

Two honest caveats. A full reset from this screen (F4 ALL) deliberately discards the earlier state and builds a fresh store; that is its job. And across firmware

releases, Free85 does not promise internal-state compatibility: an upgrade may clear the calculator's stored data, so treat firmware upgrades like the reset they can be, and copy down anything irreplaceable first.

CHAPTER

19

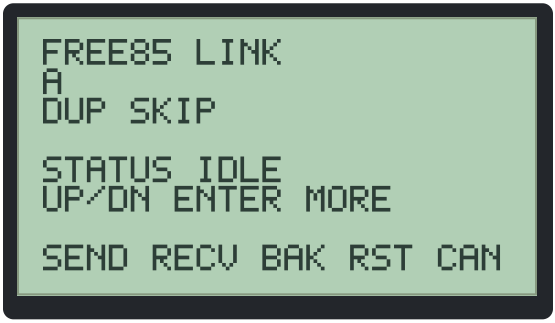
Calculator Linking

The calculator's link port is the socket on its edge that connects two machines, and with Free85 2.10 it earns a working screen: stored objects can be marked and sent to a partner calculator, received from one under a duplicate policy you choose, and the whole machine can be backed up and restored over the cable, transactionally. This chapter covers the screen and its vocabulary; everything quoted was exercised in the emulator, where complete transfers run between two emulated machines.

The link screen

19-1

The [x-VAR] key's shifted function is `LINK`. Press [2nd] [x-VAR] and the link screen opens; the `LNK` soft key in the memory browser ([F5], Chapter 18: Memory Management) leads to the same place, standing on whatever object the browser had selected:



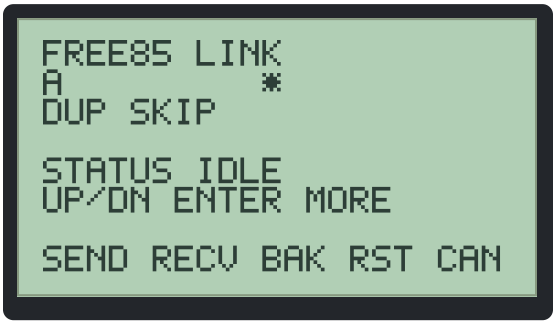
The link screen

The banner `FREE85 LINK` names the screen. Below it sits one object of chapter 18's directory at a time, `A` on a fresh machine, then the duplicate policy line `DUP SKIP`, the status line `STATUS IDLE`, the hint `UP/DN ENTER MORE`, and the soft

labels SEND RECV BAK RST CAN. The screen always opens idle, and EXIT returns to the home screen.

19-2 Marking what to send

▲ and ▼ step through the directory in the browser's order, and ENTER marks the shown object for transfer, answering with a ∗ on the right of its line; ENTER again unmarks it:



The link screen with an object marked

Mark as many objects as you like, visiting each in turn. The marks are kept in the store itself, so they survive stepping away, leaving the screen entirely, and coming back; a transfer sends every marked object in one go.

19-3 The duplicate policy

The DUP line answers the collision question before it is asked: what should a receiving calculator do when an incoming object carries a name it already holds? MORE cycles the policy from SKIP to OVERWRITE to RENAME and around again, and the setting belongs to the receiving side:

- **SKIP** keeps the receiver's object and drops the incoming copy.
- **OVERWRITE** replaces the receiver's object with the incoming one.
- **RENAME** keeps both: the incoming copy is stored under its name with the first unused letter added, so a second RATE arrives as RATEA and a third as RATEB.

19-4 Sending and receiving

F1 (SEND) posts every marked object for transfer and the status line answers STATUS WAITING: the calculator is offering the objects and waiting for a partner

that is ready to take them. F2 (RECV) waits the complementary way, ready to receive whatever a partner offers, under the DUP policy on show. When the two sides meet, the transfer runs and the receiver finishes at STATUS COMPLETE with the objects stored, named, and immediately usable. The status line's whole vocabulary is IDLE, WAITING, ACTIVE while a transfer is running, COMPLETE, CANCELLED, and ERROR.

The transfer is checked as it arrives, and it is all or nothing: a damaged transfer, or one that will not fit under chapter 18's capacity rules, is refused whole at STATUS ERROR, and an interrupted one stops at STATUS CANCELLED; either way the receiving calculator keeps exactly what it had rather than a half-applied copy.

F5 (CAN) cancels: the posted command is withdrawn and the status answers STATUS CANCELLED. Leaving the screen with EXIT in the middle of a running transfer cancels it the same way.

Backup and restore

19-5

The last pair of soft keys moves whole machines rather than chosen objects. F3 (BAK) offers a backup of everything the calculator stores, and F4 (RST) asks a partner for one; both post their request and wait at STATUS WAITING like a send. A restore replaces the receiving calculator's stored state with the backup's, and its previous objects go, so treat RST like the reset it is.

The exchange is transactional: the archive is checked whole, and a damaged backup restores nothing at all, leaving the machine exactly as it was rather than half-restored.

> **HARDWARE**
>
> marking, policies, statuses, and cancellation are quoted from the emulator's screen, and complete transfers, backups, and restores run between two emulated machines; physical hardware validation is reported separately.

The clean-room boundary

19-6

Because Free85 is written from scratch, its link protocol is its own: it does not read or write TI file formats, token streams, backup images, or ROM-level link calls, and a link partner is expected to be another Free85 machine.

Appendix A catalogues this chapter's workflow as `select-items`, `send-items`, `receive-items`, `duplicate-skip`, `duplicate-overwrite`, `duplicate-rename`, and `transfer-cancel`, and the backup work as `backup-send`, `backup-confirm`, `backup-restore`, and `backup-rollback`.

APPENDIX

A

Command and Function Catalog

Grouped by functional area. Status comes from the Free85 2.0 command ledger's status vocabulary: `equivalent` items work today; `partial` and `missing` mark tracked implementation gaps; `hardware-dependent` items await separate hardware validation; `excluded-clean-room` items are intentionally out of scope by design, not gaps. The ledger is closed: no `partial` or `missing` entries remain.

A-9 base.entry-conversion — command (equivalent)

`Bin`, `Oct`, `Dec`, `Hex`, `binary-entry`, `octal-entry`, `decimal-entry`, `hex-entry`, `->Bin`, `->Oct`, `->Dec`, `->Hex`

A-10 base.boolean — command (equivalent)

`and`, `or`, `xor`, `not`, `rotL`, `rotR`, `shftL`, `shftR`

A-11 complex.scalar — command (equivalent)

`angle`, `conj`, `imag`, `real`, `polar-complex-entry`, `->Pol`, `->Rec`, `PolarC`, `RectC`

A-12 coordinates.vector — command (equivalent)

`->Cyl`, `CylV`, `->Sph`, `SphereV`, `RectV`

A-13 list.operations — command (equivalent)

`dimL`, `->dimL`, `Fill-list`, `prod`, `seq`, `sortA`, `sortD`, `sum`, `li->vc`, `vc->li`

A-14 matrix.operations — command (equivalent)

`aug`, `cnorm`, `cond`, `det`, `dim-matrix`, `->dimM`, `eigVc`, `eigVl`, `Fill-matrix`, `Ident`, `inverse-matrix`, `LU`, `mRAdd`, `multR`, `norm-matrix`, `rAdd`, `randM`, `ref`, `rnorm`, `rref`, `rSwap`, `transpose`

A-15 vector.operations — command (equivalent)

`cross`, `dim-vector`, `->dimV`, `dot`, `Fill-vector`, `norm-vector`, `unitV`

A-16 collections.result-chaining — workflow (equivalent)

`USE R`

A-17 collections.elementwise — semantic (equivalent)

`elementwise-real`, `elementwise-complex`, `elementwise-list`, `elementwise-matrix`, `elementwise-vector`

A-18 graph.cartesian-core — command (equivalent)

`Func`, `FnOn`, `FnOff`, `Trace`, `ZIn`, `ZOut`, `ZStd`, `ZSqr`

A-19 graph.format — command (equivalent)

`AxesOff`, `AxesOn`, `CoordOff`, `CoordOn`, `DrawDot`, `DrawLine`, `GridOff`, `GridOn`, `LabelOff`, `LabelOn`, `SeqG`, `SimulG`

A-20 graph.zoom — command (equivalent)

`ZBox`, `ZDecm`, `ZFit`, `ZInt`, `ZPrev`, `ZRcl`, `ZTrig`, `zoom-factors`, `user-defined-zoom`

graph.draw — command (equivalent) A-21

`Circ`, `ClDrw`, `DrawF`, `DrInv`, `Line`, `PtChg`, `PtOff`, `PtOn`, `Shade`, `TanLn`, `Vert`, `freehand-pen`

graph.persistence — command (equivalent) A-22

`RcGDB`, `RcPic`, `StGDB`, `StPic`

graph.polar — workflow (equivalent) A-23

`Pol`, `PolarGC`, `polar-editor`, `polar-plot`, `polar-trace`, `polar-table`, `polar-analysis`

graph.parametric — workflow (equivalent) A-24

`Param`, `parametric-editor`, `parametric-plot`, `parametric-trace`, `parametric-table`, `parametric-analysis`

graph.diffeq — workflow (equivalent) A-25

`DifEq`, `dxDer1`, `dxNDer`, `diffeq-editor`, `diffeq-plot`, `diffeq-explore`, `diffeq-solve`, `diffeq-setup`, `Euler`, `Heun`, `RK4`

solver.general — command (equivalent) A-26

`Solver`, `solver-equation`, `solver-variables`, `solver-guesses`, `solver-bounds`, `solver-graph`

solver.specialist — command (equivalent) A-27

`poly`, `simult`

statistics.models — command (equivalent) A-28

`ExpR`, `LinR`, `LnR`, `P2Reg`, `P3Reg`, `P4Reg`, `PwrR`, `fcstx`, `fcsty`

statistics.commands — command (equivalent) A-29

`OneVar`, `TwoVar`, `ShwSt`, `Sortx`, `Sorty`

statistics.plots — command (equivalent) A-30

`Hist`, `Scatter`, `xyline`

strings.core — command (equivalent) A-31

`Concatenate`, `lngth`, `sub`, `Eq->St`, `St->Eq`

program.supported — instruction (equivalent) A-32

`Disp`, `Else`, `End`, `For`, `If`, `Input-number`, `Return`, `Stop`, `Then`, `While`

A-33 program.control — instruction (equivalent)

`DS<`, `Goto`, `IS>`, `Lbl`, `Menu`, `Repeat`

A-34 program.io — instruction (equivalent)

`ClLCD`, `DispG`, `getKy`, `InpSt`, `Input-string`, `Outpt`, `Pause`, `Prompt`, `PrtScrn`

A-35 program.external — instruction (hardware-dependent)

> Tracked under: `program.io`

> Target: provide an open virtual-device interface and separately report physical hardware status

`Input-CBLGET`, `Output-CBLSEND`

A-36 program.catalog — semantic (equivalent)

`all-math-from-programs`, `all-graph-from-programs`, `all-collection-from-programs`, `all-statistics-from-programs`

A-37 constants.user — workflow (equivalent)

`create-user-constant`, `edit-user-constant`, `name-user-constant`, `delete-user-constant`

A-38 characters.extended — workflow (equivalent)

`Greek-characters`, `international-characters`

A-39 memory.management — workflow (equivalent)

`memory-by-type`, `object-size`, `individual-delete`, `reset`, `leave-memory-screen`

A-40 link.transfer — workflow (hardware-dependent)

> Tracked under: `link.transfer`

> Target: separately validate the completed emulator-tested Free85 protocol on a physical cable

`select-items`, `send-items`, `receive-items`, `duplicate-rename`, `duplicate-overwrite`, `duplicate-skip`, `transfer-cancel`

A-41 link.backup — workflow (hardware-dependent)

> Tracked under: `link.backup`

> Target: separately validate transactional backup and restore on physical hardware

`backup-send`, `backup-confirm`, `backup-restore`, `backup-rollback`

clean-room.exclusions — compatibility (excluded-clean-room) A-42

> Reason: Free85 provides original user-facing equivalents without copying or promising TI binary compatibility

`TI-binary-programs`, `TI-token-streams`, `TI-file-formats`, `TI-ROM-calls`, `TI-internal-data-structures`

APPENDIX

B

Complete Key Map

Key	Normal	2nd	ALPHA
F1	F1 — Choose the first visible soft-menu item	M1 — Open memory menu slot one	—
F2	F2 — Choose the second visible soft-menu item	M2 — Open memory menu slot two	—
F3	F3 — Choose the third visible soft-menu item	M3 — Open memory menu slot three	—
F4	F4 — Choose the fourth visible soft-menu item	M4 — Open memory menu slot four	—
F5	F5 — Choose the fifth visible soft-menu item	M5 — Open memory menu slot five	—
2ND	2nd — Arm one shifted action	—	—
EXIT	EXIT — Return one navigation level	QUIT — Return to the home calculator	—
MORE	MORE — Advance to the next menu page	MODE — Open calculator mode settings	—
UP	▲ — Move selection or cursor up	—	—
DOWN	▼ — Move selection or cursor down	—	—
ALPHA	ALPHA — Arm or lock alphabetic entry	alpha — Select lowercase alphabetic entry	—
X–VAR	x-VAR — Insert the active graph variable	LINK — Select items, transfer, back up, restore, and cancel over the Free85 link protocol	x — Insert x
DEL	DEL — Delete at the editor cursor	INS — Toggle insert and overwrite mode	—
LEFT	◀ — Move selection or cursor left	—	—
RIGHT	▶ — Move selection or cursor right	—	—
GRAPH	GRAPH — Open the graph application	SOLVER — Open the general equation solver	—
STAT	STAT — Open the statistics application	SIMULT — Open the simultaneous-equation solver	—

Key	Normal	2nd	ALPHA
PRGM	PRGM — Open the program manager	POLY — Open the polynomial solver	—
CUSTOM	CUSTOM — Open the user custom menu	CATALOG — Open the callable-feature catalog	—
CLEAR	CLEAR — Clear the entry or dismiss an error	TOLER — Open numerical tolerance settings	—
LOG	LOG — Insert base-ten logarithm	10^x — Insert ten raised to a power	A — Insert A
SIN	SIN — Insert sine	SIN^-1 — Insert inverse sine	B — Insert B
COS	COS — Insert cosine	COS^-1 — Insert inverse cosine	C — Insert C
TAN	TAN — Insert tangent	TAN^-1 — Insert inverse tangent	D — Insert D
^	^ — Insert the power operator	π — Insert pi	E — Insert E
LN	LN — Insert natural logarithm	e^x — Insert natural exponential	F — Insert F
EE	EE — Insert a scientific-notation exponent	x^-1 — Insert reciprocal	G — Insert G
(	(— Insert an opening parenthesis	[— Insert an opening collection bracket	H — Insert H
)	) — Insert a closing parenthesis	] — Insert a closing collection bracket	I — Insert I
/	÷ — Insert division	CALC — Open numerical calculus tools	J — Insert J
X^2	x² — Square the current value	√ — Insert square root	K — Insert K
7	7 — Insert digit seven	MATRX — Open matrix tools	L — Insert L
8	8 — Insert digit eight	VECTR — Open vector tools	M — Insert M
9	9 — Insert digit nine	CPLX — Open complex-number tools	N — Insert N
*	× — Insert multiplication	MATH — Open the general mathematics menu	O — Insert O
,	, — Insert an argument separator	∠ — Insert the polar-angle operator	P — Insert P
4	4 — Insert digit four	CONS — Open built-in and user constants	Q — Insert Q
5	5 — Insert digit five	CONV — Open unit conversions	R — Insert R
6	6 — Insert digit six	STRNG — Open string operations	S — Insert S
-	- — Insert subtraction	LIST — Open list tools	T — Insert T
STO	STO▶ — Store a value in a named variable	RCL — Recall a named variable	—
1	1 — Insert digit one	BASE — Open number-base tools	U — Insert U
2	2 — Insert digit two	TEST — Open comparison and logical tests	V — Insert V

Key	Normal	2nd	ALPHA
3	3 — Insert digit three	VARS — Open the variable browser	W — Insert W
+	+ — Insert addition	MEM — Open memory management	X — Insert X
ON	ON — Wake the calculator or interrupt an operation	OFF — Shut down the calculator	—
0	0 — Insert digit zero	CHAR — Open the character palette	Y — Insert Y
.	. — Insert a decimal point	: — Insert a program statement separator	Z — Insert Z
(–)	(-) — Insert unary minus	ANS — Insert the previous answer	—
ENTER	ENTER — Confirm or evaluate	ENTRY — Recall the previous entry	—

APPENDIX

C

System Variables and Error Messages

The first half of this appendix collects in one place the names the calculator keeps for you; the second half is the error reference promised in chapter 1: every message the calculator can answer with, what causes it, and the way back. Every screen quoted below was produced on the machine by pressing the keys described.

System variables

- **ANS** (2nd (-)) always names the most recent numeric result. It is maintained by the calculator and cannot be stored to: `5→ANS` answers `SYNTAX ERROR`. Chapters 1 and 2 cover it.
- **A through Z** are the twenty-six named variables, present from first boot with the value `0`. They can be cleared but never removed, and every one you have never stored to reads as `0`. Chapter 2 covers storing and recalling; chapter 18 covers clearing.
- **x and X** are one variable, the graph variable: the x-VAR key types `X` in one press, both spellings read and store the same value, and `x` is the only lowercase letter accepted in a name. Chapters 2 and 4 cover it.
- **M1 through M5** are the five numeric memories on 2nd F1 through 2nd F5: with an expression on the entry line the key stores, with an empty line it recalls. Chapter 2 covers them.
- **Y1 through Y3** are the three graph function slots: GRAPH saves the home entry line into the active slot, `Y1` is active on a fresh boot, and the stored equations persist between plots. Chapter 4 covers the slots and switching between them.
- **The editor registers A, B, and R** belong to the string, complex, list, matrix, and vector editors: each editor keeps its own pair of working registers and a result register, separate from the named variables, and their contents survive

leaving the editor. Chapters 9, 11, 12, and 13 cover them, and chapter 15's statistics editor keeps its `X` and `Y` data columns the same way.

C-2 How errors present

Errors and confirmations share one full-screen dialog, introduced in chapter 1. The status line stays put, and the body shows three lines: the message name, the hint `CLEAR OR EXIT` beneath it, and `EXIT BACK` at the bottom. Pressing **CLEAR** or **EXIT** returns you to the home screen. When the error came from evaluating the home entry line, the entry is preserved with the cursor at the end, so you can fix the mistake instead of retyping it; when it came from inside an editor, the editor keeps its contents, and reopening it puts you back where you were. The statistics screens of chapter 15 deserve one extra sentence here: their result screens leave with **EXIT** for the home screen and **CLEAR** for the editor, but their guard dialogs dismiss to the home screen with either key, so press **STAT** to return to the data. Unless an entry below says otherwise, every message in this appendix presents and dismisses exactly this way.

A few screens carry their messages themselves rather than raising the dialog: the simultaneous solver's verdicts, the program run screen's stop and error reports, and the catalog's assignment confirmation. Their entries below describe their own mechanics. The link screen belongs to this family too: `STATUS ERROR` and `STATUS CANCELLED` are readings of its `STATUS` line, not dialogs, and chapter 19 walks that line's whole vocabulary, so they get no entries of their own here.

C-3 Entry and editing notices

These guard the home entry line and cost you nothing: dismiss them and your entry is exactly as you left it.

- **`ENTRY FULL`**: the entry line holds 48 characters, and the press that would add a 49th answers this instead. Chapter 1 covers the entry line.
- **`ENTRY EMPTY`**: **CLEAR** with nothing on the entry line.
- **`START OF ENTRY`**: ◀ or **DEL** with the cursor at the start of the entry.
- **`END OF ENTRY`**: ▶ with the cursor at the end.
- **`NO MORE HISTORY`**: ▲ and ▼ step through the previous entries on the home screen, and the notice answers a step with nothing to show: ▲ past the oldest entry, ▼ beyond the blank line at the newest end, or either key on a

fresh machine with no history yet. Chapter 1 covers both recall routes, 2nd ENTER and the arrow keys.

- **ALREADY AT HOME**: EXIT pressed on the home screen, where there is nowhere further up to go.
- **ALREADY AWAKE**: ON pressed while the calculator is already running; chapter 1 notes it does no harm.

Arithmetic and evaluation errors

C-4

- **SYNTAX ERROR**: the expression could not be read. The causes are as varied as typing: a malformed or incomplete expression, a chained comparison such as `2<3<1` (chapter 3), a function given the wrong number of arguments such as `MIN(1,2,3)` (chapter 3), storing to an invalid name such as `AB` or `ANS` (chapter 2), or a calculus command such as `EVAL(` before a plot has run through once (chapters 3 and 4). It means the expression could not be *read*; a value that cannot be *computed* has its own message below, and the two should not be confused when reading an old listing.
- **DIVIDE BY ZERO**: division by zero, chapter 1's specimen error. `1/0` answers it, so do `MOD(5,0)` (chapter 3) and dividing by a complex zero in the complex editor (chapter 11).
- **DOMAIN ERROR**: an argument outside a function's domain. Chapter 3 collects the home-screen causes (`LN(0)`, `ASIN(2)`, `ACOSH(0.5)`, `ROOT(−8,3)`, `FACT(−1)` and kin), chapter 11 adds `SQRT(−9)` on the real line, chapter 10 adds the word-model violations such as `ROL(1,16)` and `AND(2.5,1)`, and chapter 16 adds a `FOR` bound that is not a whole number, is outside the signed 16-bit range, or a step of zero.
- **NUMERIC OVERFLOW**: a result beyond the numeric range, whose exponents run to 127: `FACT(70)` (chapter 3), `1E99*1E99`, or a base literal beyond sixteen bits such as `0x10000` (chapter 10).
- **PRECISION LOST**: the argument is too large for its phase to be worth reporting. `SIN`, `COS` and `TAN` are supported through one million radians or one hundred million degrees; past that a fourteen-digit input no longer pins the angle down, so `SIN(1.1E6)` stops here rather than returning a plausible number (chapter 3).
- **NO CONVERGENCE**: the work budget ran out before successive estimates agreed. `FNINT(` compares 32-, 64- and 128-panel Simpson estimates and stops here

when they will not settle (chapter 3); differential-equation mode reports it when a solution runs away faster than the window can follow (chapter 7).

- **`RECURSION ERROR`**: a graph slot reached itself. One nested graph evaluation is available, so a slot may read another slot, but `NDER(1,X)` stored in slot 1, or a cycle between two slots, stops here while the unrelated slots plot as usual (chapter 4).
- **`SIGNED 16-BIT INT`**: the number-base screen asked to display a value the 16-bit word cannot hold; `2.5` and `32768` both stop here. Chapter 10 covers the word model.

C-5 Data and editor errors

- **`INVALID NUMBER`**: a number was needed and not found. The editors answer it for an entry that does not parse (a bare `.`, say, in the polynomial, simultaneous, or statistics editors; chapters 14 and 15), the string tools answer it for `S2N` on text that is not a number (chapter 9), and the list editor answers it for `DIV` where list `B` holds a zero (chapter 12).
- **`STRING TOO LONG`**: a string register holds up to 31 characters; typing a 32nd answers this, and so does a `CAT` concatenation whose combined length would not fit. Chapter 9 covers the registers.
- **`DIMENSION ERROR`**: shapes that do not fit together: lists of different sizes combined element by element (chapter 12), matrix shapes that do not match the operation (chapter 13), or the vector cross product `CRS` with two-component vectors (chapter 13).
- **`SINGULAR MATRIX`**: inverting or solving with a matrix whose determinant is zero, such as the matrix 1, 2, 2, 4. Chapter 13 covers it, and chapter 14's simultaneous solver reports the same situation with the two messages below.
- **`ZERO VECTOR`**: normalising a vector of zeros, which has no direction to keep. Chapter 13 covers the vector tools.
- **`CONSTANT ERROR`**: the user-constants screen's refusal (chapter 8): confirming an empty name with **ENTER**, saving a value under a name that is not one to seven letters, or renaming onto a name already taken. Dismissal leaves for the home screen.

Solver messages

C-6

- **NO NUMERIC RESULT**: a graph-screen numeric search found nothing to report: the root finder and its companion soft keys when the search fails (chapter 4), including on a polar radius or a differential-equation slope that never crosses zero (chapters 5 and 7). Earlier firmware also answered it from the general solver; the 2.10 solver workspace reports through its own four notices below instead.
- **ENTER EQUATION HOME**, **LOWER MUST BE < UPP**, **EQUATION DOMAIN ERR**, and **NO BOUNDED ROOT**: the four notices guarding `SOLV` in chapter 14's general solver. The first answers `SOLV` with no stored equation; type one on the home entry line and press 2nd GRAPH to store it. The second answers bounds out of order, its last letters clipped by the screen; read it as lower must be less than upper. The third answers a stored equation that cannot be evaluated across the bounds, such as `LN(X)-1` over the default `-10` to `10`, clipped the same way from equation domain error. The fourth answers an equation with no sign change between the bounds, such as `X^2+1`. Each shows the usual dialog and dismisses to the home screen with the workspace kept, so 2nd GRAPH reopens it where you left off.
- **LEADING COEFF ZERO**: the polynomial solver run with a zero leading coefficient, which would really be a polynomial of lower degree. Chapter 14 covers the editor.
- **UNIQUE SOLUTION**, **NO SOLUTION**, and **UNDERDETERMINED**: the simultaneous solver's three verdicts, shown on its own result screen under the `SIMULTANEOUS` banner with `EXIT BACK` as the only footer; the screen answers only to EXIT, and the editor reopens with every cell kept. `UNIQUE SOLUTION` heads the list of unknowns, contradictory equations answer `NO SOLUTION`, and dependent equations answer `UNDERDETERMINED`. Chapter 14 covers all three.

Statistics messages

C-7

These guard chapter 15's fitting and forecasting keys. Each shows the usual dialog, and either key dismisses to the home screen with the data kept, so press STAT to return to the editor.

- **NEED TWO SAMPLES**: a polynomial regression fit with fewer pairs than it needs, one per coefficient; the wording stays the same however many samples the

model really wanted. Chapter 15 covers the regression families.

- **POSITIVE DATA NEEDE**: a logarithmic, exponential, or power fit over data its transform cannot take the logarithm of: LNR with a zero or negative X entry, EXPR with one in Y, or PWR with either. The final letter of positive data needed falls off the 21-column screen.
- **ZERO VARIANCE**: a polynomial fit (P2 through P4) over a constant X column, which gives the least-squares system nothing to work with. The other families answer a coefficient screen rather than the notice: chapter 15 shows a constant column answering A 0 and B 0 under LIN.
- **FCSTX NEEDS 2–COEFF**: an inverse forecast (FCX) on a polynomial model when no x between the data's smallest and largest X produces the target y; an inverse forecast outside the data's range needs the two-coefficient families.

C-8 Programming messages

- **NO PROGRAM**: RUN on an empty program slot. Chapter 16 covers the program list.
- **BAD NAME**: a rename to an empty or overlong name; program names hold up to seven characters. Chapter 16 covers renaming.
- **ERROR LINE**: shown on the program run screen, not the dialog, with the failing line's number after it, as in ERROR LINE 2: a line the runner cannot make sense of, an INPUT entry that does not parse, a CALL to an empty slot, or a nesting bound exceeded. Fix the line in the editor and run again; a failed run never alters the source. Chapter 16 covers running and its limits.
- **STOPPED LINE**: the run screen's report that you stopped the program, with the line it was on, as in STOPPED LINE1; ON, EXIT, and CLEAR all stop a run this way. Chapter 16 covers stopping.

Two program screens look like interruptions but are suspensions, not errors: a run waiting on you, not a run gone wrong. A MENU line suspends the run on a chooser whose banner reads PROGRAM MENU and whose footer reads F1–F5 SELECT, and a PAUSE line suspends it on a screen reading PAUSED over PRESS A KEY. A soft key picks a menu entry and any key resumes a pause, except ON, EXIT, and CLEAR, which stop the run from either screen. Chapter 16 covers both, along with the INPUT, PROMPT, and INPST entry screens that suspend a run the same way.

Confirmations

C-9

Good news arrives in the same dialog as bad:

- **`MEMORY STORED`**: a numeric memory key (2nd F1 through 2nd F5) evaluated your entry and stored the result; the entry is intact after dismissal. If the entry does not evaluate, the answer is **`MEMORY ERROR`** and nothing is stored. Chapter 2 covers the memories.
- **`VARIABLES CLEARED`**: the memory browser's `VAR` key cleared `A` through `Z` to `0`. Chapter 18 covers the bulk clears.
- **`PROGRAMS CLEARED`**: the browser's `PGM` key emptied the program storage. Chapter 18 again.
- **`TOLERANCE CHANGED`**: 2nd CLEAR cycled the numeric tolerance through `1E-6`, `1E-8`, and `1E-10`, confirming each press. Chapter 3 covers the tolerance.
- **`ASSIGNED F2`** and its siblings `ASSIGNED F1` through `ASSIGNED F5`: the catalog's confirmation that a function was assigned to a custom-menu slot, shown on the catalog screen itself rather than as a dialog. Chapter 1 covers the custom menu.

Messages you are unlikely to meet

C-10

The calculator defines a few messages that no ordinary key sequence produces. `NO OBJECTS` is the memory browser's answer to an empty store, and the store is never empty (chapter 18). `EVALUATOR NEXT`, `NO ALPHA MAP`, and `FEATURE PLANNED` round out the set; we found no key sequence that shows any of them. Earlier editions listed `ZERO VARIANCE` here too; the 2.10 polynomial fits made it reachable, and it now has its entry among the statistics messages above. If one of the remaining four ever greets you, treat it as this book's cue for an update.

APPENDIX

D

Feature Status

Chapter status

Chapter	Topic	Status
1	operation, modes, editing and previous entries	equivalent
2	variables and stored data	equivalent
3	mathematics, calculus and comparisons	equivalent
4	Cartesian graphing, drawing, formats and persistence	equivalent
5	polar graphing	equivalent
6	parametric graphing	equivalent
7	differential-equation graphing	equivalent
8	physical and user constants plus conversions	equivalent
9	strings and characters	equivalent
10	number bases and Boolean operations	equivalent
11	complex numbers	equivalent
12	lists	equivalent
13	matrices and vectors	equivalent
14	equation, polynomial and simultaneous solving	equivalent
15	statistics and statistical plots	equivalent

Chapter	Topic	Status
16	calculator programming	equivalent
17	worked application examples	equivalent
18	memory management	equivalent
19	calculator linking	hardware-dependent — Free85 item transfer and backup are complete in fault-injected emulator loopback; physical-cable validation is reported separately
appendices	functions, system variables and errors	equivalent

D-2 Remaining non-equivalent areas

Every non-`equivalent` entry from the Free85 2.0 parity gap report (spec/free85/v2-parity-gaps.yaml), sorted by work package owner. `equivalent` areas are complete and are omitted here (see Appendix A for the full catalog).

- **link.transfer** [hardware-dependent, owner `14.9`, area link]: physical-cable validation remains separate from complete emulator-tested Free85 transfer workflows
- **link.backup** [hardware-dependent, owner `14.9`, area link]: physical-cable validation remains separate from transactional emulator backup and restore

FREE85

The Free85 Guidebook · For firmware 3.0 · Second Edition

Free85 3.0 · Typeset from the Markdown sources with pandoc and Paged.js, and rendered to PDF by headless Chromium. Set in Charter, Helvetica Neue, and Menlo.

https://chriswilson2020.github.io/Free85/

F R E E 8 5

chriswilson2020.github.io/Free85