HP-15C Quick Reference

Memory & Display

Memory	Approx. 462 bytes of memory corresponding to 66 registers, 7 bytes each, 4-level stack, Last-X, index register I. Nonvolatile memory, mostly merged program commands (1 byte per instruction)
Pr Error	Displayed when the contents of the nonvolatile (continuous memory) has been lost
Number separator	Turn off, press & hold ON, press ".", release ON, release "." This toggles between using a dot or comma for the decimal separator.
Global reset	Turn off, press & hold ON, press "-", release ON, release "-" This clears all permanent memory!
MEM	Displays memory assignment in the form "RR UU pp – n" with: RR: Number of highest storage register. At least 1 which means that R0, R1 and the index register I are always present. Register 0-19 correspond to 0-9 & .0-.9 and can be accessed directly by STO/RCL. Higher registers can be reached thru indirect addressing only. UU: Number of uncommitted registers. Use DIM to commit them to storage registers. Uncommitted registers are automatically converted to program space when needed. pp: Number of registers containing program instructions. One register consists of 7 bytes and can hold 7 program steps (except for a few instructions that occupy two bytes). n: Number of bytes left before next uncommitted register is converted to program space. In total there are 66 registers corresponding to 462 bytes. The initial setup is "19 46 0-0": 20 storage registers (0-19), 46 uncommitted registers, corresponding to approx. 322 program steps.
DIM (i)	Use this command to select the number of registers committed to storage. The argument must be passed in X. It specifies the highest storage register number. Registers containing program instructions cannot be converted to storage registers! X must be at least 1 so there will always be R0 and R1 available. The maximum is 65
FIX 0-9	Select fix-point format
SCI 0-9	Select scientific format with exponent
ENG 0-9	Select engineering format with exponent always being a multiple of 3

Clearing Data

←	RUN mode: Deletes either the last digit during number entry or the entire X-register in case number entry has been terminated. PRGM mode: Delete the currently displayed program step
CLEAR ∑	Clear stack and summation registers 0-5
CLEAR PRGM	RUN mode: Set program counter to 000 PRGM mode: Erase entire program memory
CLEAR REG	Clear all storage registers
CLEAR PREFIX	Clear prefix key and briefly display all 10 digits of the mantissa
CL X	RUN mode: Clear X-register PRGM mode: Store the CLX command as a program command

Storage Registers & Indirect Addressing

STO 0-9, .0-.9	Store X in the specified storage register. By default, 20 registers are available
STO + 0-9, .0-.9 STO – 0-9, .0-.9 STO x 0-9, .0-.9 STO ÷ 0-9, .0-.9	Register store arithmetic: Register OP X → Register.
RCL 0-9, .0-.9	Recall number from storage register to X-register
RCL + 0-9, .0-.9 RCL – 0-9, .0-.9 RCL x 0-9, .0-.9 RCL ÷ 0-9, .0-.9	Register recall arithmetic: X OP Register → X.
X↔ 0-9, .0-.9	Exchange X with one of the storage registers
STO I	Store X in index register
STO +–x÷ I	Register store arithmetic with index reister
RCL I	Recall value from index register
RCL +–x÷ I	Register recall arithmetic with index reister
X↔ I	Exchange X with index register
STO (i)	Store X in the register pointed to by I. Values of I and corresponding registers: `0-9 → R0-R9, 10-19 → R.0-R.9, 10 → I`
STO +–x÷ (i)	Perform indirect register storage arithmetic
RCL (i)	Recall value from the register pointed to by I
X↔ (i)	Exchange X with the register pointed to by I
FIX I, SCI I, ENG I	Use the index register to specify the number of digits
RCL ∑+	Recall ∑x and ∑y from the summation registers into X & Y
LST X	Recall last value of X-register as it was before the previous operation
STO A-E	Used to enter elements in matrices, see **Matrix Operations**

Functions (Selection)

RAN#	Create random number $0 \leq X < 1$
STO f RAN#	Store X as the new random number seed
$\rightarrow$ P	Convert (X=x,Y=x) from orthogonal to polar coordinates (X=r,Y=θ) See label on the back of the calculator
$\rightarrow$ R	Convert (X=r,Y=θ) from polar to orthogonal coordinates (X=x,Y=x)
$\rightarrow$ H.MS	Convert fractional hours to hours, minutes & seconds: `H.MMSSs`
$\rightarrow$ H	Convert hours, minutes & seconds `H.MMSSs` to fractional hours
$\rightarrow$ RAD	Convert degress (360) to radians (2π)
$\rightarrow$ DEG	Convert radians (2π) to degress (360)
Py,x	Permutations = Y! / (Y-X)! Number of possibilities to select X elements from a group of Y different elements where different sequences count separately.
Cy,x	Combinations = Y! / [X! • (Y-X)!] Number of possibilities to select X elements from a group of Y different elements where different sequences *do not* count separately.
x!	Faculty and Gamma. Calculates $\Gamma(x+1)=n!$ for positive and non-integer negative numbers
RND	Rounds X to the number of currently displayed digits
FRAC	Returns the fractional part of X
INT	Returns the integer part of X
y^x	Y to the power of X. Works also for negative Y in case X is integer
%	Calculates X percent of Y. Does not pop the stack!
Δ%	Percential difference from Y to X. Does not pop the stack!

Trigonometric Functions

DEG	Set trig mode "degrees" (360)		
RAD	Set trig mode "radians" (2π), indicated in display		
GRD	Set trig mode "grad" (400) , indicated in display		
SIN	COS	TAN	Trigonometric functions, performed in current mode (DEG, RAD, GRD)
SIN^{-1}	COS^{-1}	TAN^{-1}	Inverse trig functions
HYP SIN	HYP COS	HYP TAN	Hyperbolic functions (independent of trig mode!)
HYP^{-1} SIN	HYP^{-1} COS	HYP^{-1} TAN	Inverse hyperbolic functions

Summation & Statistics

General	The statistics registers occupy the storage registers 2-7, see calculator's back label. See section **Clearing Data** for statistics register deletion. Stats registers can also be used for vector addition and substraction! Register usage: 2=n, 3=$\sum x$, 4=$\sum x^2$, 5=$\sum y$, 6=$\sum y^2$, 7=$\sum xy$
$\sum$+ STO $\sum$+	Add X and Y to the stats registers. This will display the total number of entries and disable stack lift so that the next entry will overwrite the count.
$\sum$-	Substract X and Y from the stats registers
RCL $\sum$+	Recall $\sum x$ and $\sum y$ from the summation registers into X & Y

$\overline{x}$	Calculate $\sum x$ & $\sum y$ mean value and place result in X & Y. Requires n>0
s	Calculate $\sum x$ & $\sum y$ standard deviation and place result in X & Y. sx = SQRT[{n$\sum x^2$ – $(\sum x)^2$} / {n(n-1)}], accordingly for sy. Requires n>1
L.R.	Linear regression. Calculates a straight line thru the (X,Y) data points and returns the slope of the line in Y and the y-offset in X. Requires n>1
$\overline{y}$,r	This function assumes a straight line thru the (X,Y) data points and calculates for a given X the approximated $\overline{y}$ value which is returned in X. In Y this function returns an estimate how close the data points come to a straight line. +1 indicates that all points lie on a line with positive slope, -1 indicates that all points lie on a line with negative slope, 0 indicates that an approximation by a straight line isn't possible. Requires n>1

Programming

P/R	Toggles between RUN (program execution) and PRGM (program entry) mode. See section **Clearing Data** for program memory and program step deletion.
SST	RUN: Display and execute next program step PRGM: Step forward thru program, scolls when held down
BST	RUN: Display and go back to previous program step but do not execute any program code PRGM: Step backwards thru program, scolls when held down
Inserting steps	Program entry starts with line number 1. Line "000-" indicates the start of the program space. Commands are inserted after the currently displayed line. Program code values indicate the row & column of a command with the exception that numbers are displayed as such. Prefix keys have their own code. Example: 001-42.21. 0 corresponds to "LBL 1" (42=f, 21=SST/LBL, 0=0)
f A-E	RUN: Execute program starting at the given label. An error occurs if the label is not found. Any keypress will halt the program! PRGM: Insert a "GSB label" command
USER	Normally, "f A-E" must be pressed to execute a program, see above. In USER mode the prefix-f is not needed, ie. pressing e^x will immediately execute the program starting at label B. Use the prefix-f to reach the key's normal function. USER mode is indicated in the display
R/S	RUN: Continue program at current program counter PRGM: Insert R/S command which will halt the program at this location
RTN	RUN: Set program counter to 000 PRGM: Insert a RTN instruction. This will return from a subroutine or at the top level end the program and set the program counter to 000
GTO CHS nnn	RUN & PRGM mode: Jump to program line nnn

LBL 0-9, .0-.9, A-E	Insert label
GT0 0-9, .0-.9, A-E	RUN: Set program counter to the specified label PRGM: Insert a GTO instruction
GSB 0-9, .0-.9, A-E	RUN: Execute the program starting at the given label PRGM: Insert a GSB instruction. A maximum of *seven* subroutine calls can be nested
Flags	There are 10 flags, 0-7 are user flags. Flag 8 & 9: 8: Complex flag. Automatically set when complex mode is activated. To deactivate complex mode explicitly clear this flag. Indicated by "C" in the display. See section **Complex Numbers** 9: Overflow flag. Automatically set by an overflow condition (result ≥1E100). Causes the display to blink. If the overflow occurs during program execution the program continues using a value of 9.99...E99 and the display blinks when the program finally stops. Cleared by CF9 or pressing "←". Can be used to provide program-controlled visual feedback. SF n: Set flag n, CF n: Clear flag n F? n: Execute next step if flag is set, skip next step if flag is clear
TEST comparisns	Only two comparisn are directly available on the keyboard: X≤Y, and X=0 Others must be entered using the TEST n command: 0: X≠0 1: X>0 2: X<0 3: X≥0 4: X≤0 5: X=Y 6: X≠Y 7: X>Y 8: X<Y 9: X≥Y If camparisn is `false`: Skip the next program step If camparisn is `true` : Execute the next program step
ISG 0-9, .0-.9, I	Increment and skip if greater. This loop command uses the specified register which must contain a value in the form `nnnnn.xxxyy` where: ±nnnnn: Current (initial) loop counter value xxx: Comparisn value for loop counter yy: Loop counter increment (or decrement for DSE), if y=0 then 1 is used instead ISG first increments n by y and then compares the new n to x: If n>x the next program step is skipped If n≤x the next program step is executed Ie. if initially I=0.023 then the loop will run from 0 to 22 (or 1 to 23)
DSE 0-9, .0-.9, I	Decrement and skip if equal (or smaller). DSE first decrements n by y and then compares the new n to x: If n≤x the next program step is skipped If n>x the next program step is executed
GTO I	Jump to the label indicated by the I register. Only the integer part of I will be used! Values of I and associated labels: I≥0: 0...9 → LBL 0...LBL 9, 10...14 → LBL A...LBL E I<0: Jump to the line number indicated by the absolute value of I. Ie. if I=–5.3 the jump will go to line number 5.
GSB I	Perform subroutine call to the label indicated by the I register
PSE	Halt program for about 1 second and display the X-register

Complex Numbers

Memory	In complex mode a complex stack including Last-X register exists. The needed five registers are allocated from the uncommitted memory space, see MEM.
f I *-or-* Re↔Im	Automatically turns on the complex mode. Indicated by "C" in the display. To turn off complex mode clear flag 8 (CF8). **NOTE**: If stack lift is enabled and a number is keyed in, a stack lift occurs and the imaginary part is set to 0!
Real number	If stack lift is enabled: Enter real part
Imaginary number	If stack lift is enabled: Enter real part, press Re↔Im
f I	Complex number input: <real part> ENTER <imaginary part> f I
f (i)	Display imaginary part of number while (i) is held down
Re↔Im	Exchange real and imaginary part
CHS	Changes sign of real part only! Use Re↔Im to negate the imaginary part as well
CLx or ←	Clears only the real part. However, this disables stack lift for both the real and imaginary stack so the entry of a complex number after "←" will do the expected thing
STO & RCL	STO & RCL only act on the real part of the number! Store: STO 1, Re↔Im, STO 2, Re↔Im Recall: RCL 2, RCL 1, f I *-or-* RCL 2, Re↔Im, ←, RCL 1 (this does not disturb the stack)
x↔y	Replace both real and imaginary part of X and Y register
R↓ R↑	Shift both the real and imaginary part
Sqrt x^2 Ln Log 1/x e^x hyp sin cos tan hyp^{-1} sin cos tan	All these unary functions work in complex mode as well. **NOTE**: To calculate sqrt(-1) the complex mode must be already enabled or otherwise an error occurs!
ABS	Calculates magnitude of complex number
+ - x ÷ y^x	All these binary functions work in complex mode as well
sin cos tan sin^{-1} cos^{-1} tan^{-1}	Trigonometric functions are only executed in radians (2π)
→ P	Convert from rectangular coordinates (real=X, imaginary=Y) to polar coordinates (real=R, imaginary=θ). This operation is affected by the current trigonometric setting (DEG,RAD, GRD)
→ R	Convert from polar coordinates (real=R, imaginary=θ) to rectangular coordinates (real=X, imaginary=Y). This operation is affected by the current trigonometric setting (DEG,RAD, GRD)
Conditional tests	These tests work for complex numbers and operate on both the real and imaginary part: x=y, TEST 0 (X≠0), TEST 5 (X=Y), TEST 6 (X≠Y) All other tests ignore the imaginary part of the complex number

Matrix Operations

Memory	A total of 64 matrix elements can be used in a total of 5 matrices named A-E. Different matrices can have different size; sometimes the result of a matrix operation can overwrite the input matrix. The registers for the matrix elements are allocated from the uncommitted registers space, see MEM. See further down for **complex matrices**.
MATRIX 0	Redimensions all matrices to 0x0 thus freeing up all memory occupied by matrices
Matrix descriptors	The stack registers, Last-X and index register I as well as ordinary storage registers can contain "*matrix descriptors*" which refer to one of the matrices A-E. Ie. if there are two matrix descriptors in X and Y then pressing "+" will add them and put the result in the *result matrix.* Matrix descriptors can be moved around in the stack and to/from storage registers like ordinary numbers
DIM A-E	Dimensions one of the matrices A-E. It will have as many rows as specified in Y and as many columns as specified in X. Whan an existing matrix is redimensioned values are lost or zeros inserted. Refer to pg. 142 of the Owner's Handbook
DIM (i)	If I contains a matrix descriptor then the DIM operation will be performed on the matrix specified in I. This indirect method applies to other matrix operations, see below.
RCL DIM A-E, (i)	Places the matrix' dimensions in X and Y. A non-exisiting matrix has dimensions 0x0
RCL MATRIX A-E	Put a matrix descriptor in the X register. This displays the matrix' name and its dimensions
STO 0-9, .0-.9, I RCL 0-9, .0-.9, I	Matrix descriptors can be stored in and recalled from ordinary storage registers
MATRIX 1	Stores 1 in R0 and R1 which are used to index matrix elements. Useful in preparation of matrix element input
STO A-E, (i) RCL A-E, (i)	Store X in the matrix element of matrix A-E which is addressed by registers R0 and R1. R0 is the row and R1 the column number, starting from 1. RCL recalls the matrix element. While the A-E key is held down, the matrix name, row and column are displayed. R1 & R0 are automatically incremented in USER mode, see below
USER	When user mode is active, a STO A-E, (i) or RCL A-E, (i) operation will automatically increment the column index in R1 until it wraps back to 1 in which case the row index R0 is increment until it wraps back to 1 as well. So in user mode *all* matrix elements can quickly be entered and recalled
STO +−x÷ A-E, (i) RCL +−x÷ A-E, (i)	Matrix element arithmentic. Does not increment R1/R0 in USER mode

STO g A-E, (i)	Same as above but the stack's Y register contains the row number and X the column number, starting from 1. The value must be present in Z. Both X & Y will be popped from the stack so that the value ends up in X.		
STO g A-E, (i) RCL g A-E, (i)	Same as above but the stack's Y register contains the row number and X the column number, starting from 1. RCL will pop X & Y from the stack and then push the matrix element into X		
STO MATRIX A-E	X is a number: Store the value of X in all matrix elements. X is a matrix: Copy matrix in X to the specified matrix. The destination matrix will be redimensioned		
RCL MATRIX A-E	Put the matrix descriptor of the specified matrix in X		
x↔ A-E, (i)	Exchange X with the matrix element of A-E specified by R1/R0. R1 & R0 are not affected		
DSE A-E, (i) ISG A-E, (i)	Decrements/increments the matrix element of A-E or (i) specified by R1/R0. R1 & R0 are not affected. See DSE & ISG in section **Programming**		
RESULT A-E	Specifies the *result matrix* (default is A). This is the matrix that will hold the result of a matrix operation. Not all operations require a result matrix. The result matrix will automatically be dimensioned so that it can properly hold the result. For some matrix operations the result matrix can be identical to one of the input matrices		
STO RESULT	When a matrix descriptor is already present in X then this matrix will be used as the result matrix		
RCL RESULT	Recalls the descriptor of the result matrix into X		
Unary matrix operations	*Result in X*	*Effect on matrix specified in X*	*Effect on RESULT matrix*
CHS	None	Changes sign of all matrix elements	None as long as X<>RESULT
1/x	Descriptor of RESULT. X must be square	None as long as X<>RESULT	Inverse of matrix X. If it is singular, then 1/x will calculate the inverse of a matrix that is close to X.
MATRIX 4	None	Replaced by transpose X^T	None as long as X<>RESULT
MATRIX 7	Row norm: Largest sum of absolute values of all rows	None	None
MATRIX 8	Frobenius or Euclidian norm of X: Square root of the sum of all matrix elements	None	None
MATRIX 9	Determinat of matrix. X must be square	None as long as X<>RESULT	LU decomposition of matrix X
Scalar matrix operations			

Operation between a matrix and a scaler (=a plain number)		
+	If X is a matrix and Y a scalar (or vice versa) the scalar will be added to each element of the matrix	
x	If X is a matrix an Y a scalar (or vice versa) each element of the matrix will be multiplied by the scalar	
	X=scalar, Y=matrix	X=matrix, Y=scalar
-	Substract scalar from each matrix element	Substracts each matrix element from scalar
÷	Divide each matrix element by scalar	Calculates the inverse of the matrix and then multiplies each matrix element with scalar
Binary matrix operations X and Y contain matrix descriptors		
+	Add X+Y $\rightarrow$ RESULT, where RESULT may be X or Y. X & Y must have the same dimensions	
-	Substract Y-X $\rightarrow$ RESULT, where RESULT may be X or Y. X & Y must have the same dimensions	
x	Multiply Y•X$\rightarrow$ RESULT, where RESULT may neither be X or Y. X & Y must have the compatible dimensions	
÷	Calculate X^{-1}•Y$\rightarrow$ RESULT, where RESULT may be Y but not X. X will be replaced by its LU decomposition. If X is singular it is replaced by a non-singular matrix close to X. Note that the order of X and Y is reversed! It corresponds to the Y/X order. X must be square and have dimensions compatible with Y	
MATRIX 5	Calculate Y^T•X$\rightarrow$ RESULT, where RESULT may neither be X nor Y. X & Y must have compatible dimension	
MATRIX 6	Calulatest the residual: RESULT – Y•X$\rightarrow$ RESULT The descriptor of RESULT is placed in X. RESULT may neither be X nor Y. X & Y must have compatible dimension	
Matrix in LU form	Its descriptor is displayed with two dashes after the matrix name A-E. Operations ÷ and determinate (MATRIX 9) calculate a LU decompsed matrix. The following operations can be performed with the LU decomposition as with the original matrix: 1/x, ÷ (X=matrix) and MATRIX 9	
Complex matrices Refer to pg. 160ff of the Owner's Manual. Complex matrix operations are not supported directly. However, these operations can be rewritten so that they can be solved using only real matrices. The HP-15C provides a number of functions to simplify the conversions between complex and corresponding real matrixes		
Py,x	Converts $X^C \rightarrow X^P$. Number of rows of X must be even	
Cy,x	Converts $X^P \rightarrow X^C$. Number of columns of X must be even	
MATRIX 2	Expand X^P to $\overline{X}$. Number of rows of X must be even	
MATRIX 3	Collapse $\overline{X}$ to X^P. Number of columns of X must be even	
GSB I, GTO I	If I contains a matrix then the natrix name A-E is used as the target label of the GSB or GTO	

X=0	Always returns false if X contains a matrix descriptor
TEST 0 (X≠0)	Always returns true if X contains a matrix descriptor
TEST 5 (X=Y)	Returns true if X and Y contain the same matrix descriptor. This does not compare any matrix elements!
TEST 6 (X≠Y)	Returns true if X and Y contain a different matrix descriptor or if X or Y doesn't contain a matrix at all
Last X	Operations which affect the RESULT matrix or produce a scalar in X also affect Last X in the usual way
Maxtrix operations in a program	
USER mode	When USER mode is on STO & RCL operations on matrix elements increment the R1/R0 register (see above). When such an instruction is entered in a program a "u" replaces the dash after the program line number to indicate that the command will increment R1/R0. If in programmed USER STO & USER RCL mode the R1/R0 registers wrap around to (1,1) the next program line is skipped. This can be helpful when accessing all matrix elements without explicit knowledge of the matrix dimensions
MATRIX 7 MATRIX 8	Row norm & Frobenius norm. Puts original X into Last X. Then if X is a matrix the norm is calculated and placed in X and the next program line is executed. If X is a scalar it remains unchanged and the next program line is skipped. This can be used to test whether X contains a matrix or a scalar

Root Finding (Solver)

Memory	The solver needs 5 registers. These are allocated from the uncommitted registers space, see MEM. The solver and the numerical integrator (see below) share their registers
SOLVE 0-9, .0-.9, A-E	Finds real root of a function. This is a value X where the function f(X) evaluates to 0. • SOLVE expects two initial guesses for X in X and Y. These values can be used to narrow down the serach for a root in case f(x) has multiple roots. X=Y is permissable • It then makes repeated GSB calls to the label with the current X value being present in the stack's X, Y, Z and T register • The program at the label must calculate the function f(X) and return the result in X before it executes the RTN • When SOLVE finally ends the stack will contain the following values: X: Value for which f(X)=0, this is the "root" Y: X value of the 2nd to last evaluation step Z: f(X) at the root value – should be 0! • If no root can be found Error 8 occurs (in RUN mode) • Note that SOLVE eats up two of the seven possible GSB levels: One for SOLVE and one for the calls to the user function • The program which calculates f(x) must not call SOLVE (no nesting)
Complex mode	SOLVE ignores the complex stack and can only calculate real roots

SOLVE in a program	If SOLVE can find a root the next program line is executed, otherwise skipped
Misc	• To speed up the root finding process rewrite your function f(x) so that it returns 0 if $\|f(x)\|<\varepsilon$. Or count the number of iterations inside the calculation of f(x) and stop when a limit has been reached • Even if no root can be found the stack registers contain the above mentioned values. These often give a hint why the root finding failed • To find multiple roots eliminate an already known root R by dividing the function by (x-R) • Fore more details see HP-15C Owner's Handbook, Appendix D, pg.220ff and The HP-15C Advanced Functions Handbook

Numerical Integration

Memory	The integrator needs 23 registers. These are allocated from the uncommitted registers space, see MEM. The integrator and the solver (see above) share their registers
$\int^x_y$ 0-9, .0-.9, A-E	Integrates function f(X) at the given label for X values running from Y to X • $\int^x_y$ makes repeated GSB calls to the specified label with the current X value being present in the stack's X, Y, Z and T register • The program at the label must calculate the function f(X) and return the result in X before it executes the RTN • When $\int^x_y$ ends the stack wil contain these values: X: The integral of f(x) Y: The uncertainty of the result: $\int^x_y$ f(x) = X±Y Z: Upper integration limit T: Lower integration limit • Note that $\int^x_y$ eats up two of the seven possible GSB levels: One for $\int^x_y$ and one for the calls to the user function • The program which calculates f(x) must not call $\int^x_y$ (no nesting). However, SOLVE and $\int^x_y$ can be nested
Accuracy	The integral is only evaluated to the accuracy specified by the current FIX, SCI or ENG format! The more digits have been specified the more accurate the integral will be – but calculating it will take longer
Misc	• Initially, $\int^x_y$ will evaluate f(x) only at a few sample points. Then the number of sample points are increased until the calculated integral doesn't change any more. This has one important consequence: The integration limits should be close to the area where the function is "interesting". Ie. exp(-x²) around x=0 – if this function is integrated from 1E-50 to 1E+50 then the result will be 0 because the algorithm missed the interesting part around 0 • Fore more details see HP-15C Owner's Handbook, Appendix E, pg.240ff and The HP-15C Advanced Functions Handbook