decb MANUAL

Welcome to decb, the 1974 DEC-BASIC interpreter. This manual, shaped as a man or info document, is aimed to inform the user about the interpreter characteristics and behaviors, but contains no BASIC language statements explanations, assuming the reader has a sufficient knowledge about BASIC programming and constructs. Please consult the documents you find in the list at the bottom of this manual, all freely downloadable, for information about the specifics of the DEC-BASIC.

DESCRIPTION

This is decb, a program for interpreting and executing DEC-BASIC programs written according to version 17 (mainly 1975-76 version 17F, a direct evolution, with minor issues, of the original 1969 Version V of the mythical Dartmouth BASIC); this version 17 ran on DEC machines in the early Seventies, over the MONITOR shell; decb is a shortcut for 'DEC BASIC'. References to documentation are in [2], [3] and [4].

The decb BASIC language is a very compliant version of the DEC-BASIC one; all the restrictions applied "there", are generally applied "here"; all features are inserted, and a BASIC program yields, apart from some minor issues, just the same results and output lines of the original DEC-BASIC.

decb owes all of its BASIC engine to vibes, a program I wrote years ago which emulated the MONITOR, the BASIC interface and a primitive version of the DEC-10 BASIC; vibes, in turn, owes many things to dib, my former emulator for 1966 Dartmouth BASIC Version III [1], from which I grabbed the parser, the basic language statements running engine, the MAT statements and the error detection system; on top of these I promoted the BASIC interpreter to level V. This list would not be complete if I did not quote here Mr. Spinellis, whose 'dds basic' interpreter served as a first engine for my dib. Thanks, Diomidis.

decb is a UNIX program, it's completely free and is distributed according to the GNU Public License version 3. Donations are welcome through Paypal to tbin at libero dot it.

SYNOPSIS

decb [options] [filename]

OPTIONS

Option List

```
−c, −−capitalize
        Enable complete capitalization
     −−conditions
        Print GPL3 redistribution conditions and exit
     −−errors−list
        Print a complete errors list and exit
−g, −−grace
        Enable grace mode in listing (spacing and indenting)
−h, −−help
        Display this information and exit
     −−insert
        Insert line numbers before each line of file and list
     −−length
        Print number of characters in the program file
−L [n1[-n2]]
            List file lines from <n1> to <n2>
     −−list
        List all file lines
−N <name>
        Show in header <name> instead of current file name
−n, −−no−header
        Disable printing of header and footer
−n, −−no−reset
        Disable resetting of variables between two CHAIN calls
−P <num>
        Set <num> as the maximum page length for current program
−R [n[,f,k]]
        Resequence and list file lines (f becomes n, step k)
     −−resequence
        Resequence and list file lines (default values for f,n,k)
     −−reverse
        Reverse listing order for option -l
−S <num>
        Run program from line number <num>
     −−save
        Set save mode for options -r and -i (enable overwriting)
−T <num>
        Set <num> as the tab stop for list indenting, in grace mode
−v, −−version
        Display version and exit
     −−warranty
        Print warranty conditions from GPL3 and exit
```

The [filename] must be a text file (in UNIX or DOS format), it may have any extension, and may be located anywhere in the system tree, provided

the whole path is specified; remember that whenever a file is called within a program (with CHAIN or FILES, for instance), the called file must be stored in the same directory, or it must be called along with the whole path. If more than one filename is specified, only the first will be executed; the others will be ignored.

Any [options] may be input as interspersed single- and double-dash unities; single-dash options may of course be grouped, provided the ones requiring an argument are left as last. Any or no blank spaces may be put between any option and its argument, or between arguments. E.g.:

```
 $ decb -cN VARS -S20 VAR12.BAS
```

Here, option -c is directly followed by option -N (since -c requires no arguments), but option -N is not directly followed by option -S (which stands on its own), because -N requires an argument. The arguments of -N and -S may or not be followed by a blank.

If you find some option weird, remember this is a program conceived to be run as a BASIC language server to vibes (the MONITOR and the BASIC interface emulator, not yet ready for publication).

Options revealed

Options are self explanatory (you can skip this, if you think you know the matter), but I guess some words won’t hurt. Apart from legal and informative options (which need no comment), here’s a survey of the other available options:

–c, ––capitalize
Usually, input by keyboard is not changed by decb: if you type a string, for instance, with lower and upper letters, it will, when processed or printed, conserve intact the original aspect.

If you want, instead, to be ’conservative’, emulating the behavior of the Early Seventies machines - which could use capital letters only - you can enable this option -c/--capitalize, which will treat all text capitalized. It will also turn to upper all text in the DATA lines and in all literal strings. This option can be type also as --all-upper and is available as a PRAGMA option.

––errors–list
This option is useful only if you want to know the error codes associated with the error messages, or to see the list of all the error messages present in the system. It can be typed also as --errors and as --errorslist. A complete list is in this document anyway. (See below). Not all the error messages in the list have been implemented, due to different issues in algorithms or in the operating systems, but all errors are listed for documental purposes.

–g, ––grace: see option -L

––insert
Occasionally, you may need to convert a program written for another

BASIC, a program, say, with no line numbers (except for the labeled addresses for GOTOs, GOSUBs, USING and the like). If it’s a long program, you’d probably spend much of your time in inserting line numbers one by one. This option does this for you, with line numbers starting from 10 and with step 10, breaking also multiple instructions on the same line, if separated by colon (:) or backslash (\) and updating inner references (GOTOs, GOSUBs, etc..); options -t, -g/--grace, --reverse and -s/--save have an effect on this option. It can be typed also as --insert-line-numbers.

Beware: this algorithm cannot work in the case of an IF..THEN followed by an instruction and not by a line reference: this must be done manually; look at the following example, where the one-line (useless) program:

```
 IF A=0 THEN PRINT "WOW!" : B=0 : C=24 : PRINT "OK!"
```

is converted by option --insert into:

```
 10 IF A=0 THEN PRINT "WOW!"
 20 B=0
 30 C=24
 40 PRINT "OK!"
 50 END              ’ END ADDED IF MISSING
```

After this operation, line 10 should be manually changed into:

```
 10 IF A<>0 THEN 50      ’ INVERT CONDITION TEST
```

(the address must be set to the line following the last statement of the IF line) and a new line should be added manually, afterwards, containing the statement after THEN, like this:

```
 15 PRINT "WOW!"
```

After all these corrections, run the -R/--resequence --save options, and your file is ready and converted.

Finally, if your program bears instructions like GOTO I*20, take a breath and start converting them manually: decb cannot help you, here, because decb (and the DEC-BASIC of course) cannot resolve formulas as number references, because the variables are instantiated at runtime, and are not known by option --insert.

––length
This command emulates the LENGTH command, returning the length of characters of the program passed to it, before exiting. The length is the exact value reported by the UNIX command ls; the result may be different from the DEC values (from the same source), due to the different File Systems involved. It can also be typed as --len.

–L [n1[-n2]], ––list
This is a very nice feature, trying to emulate the LIST command. If enabled, it will cause decb to list the program, rather than executing

it, but it will print the lines as in memory. So, no special markers or the like will appear, but all regular comments and even the void lines. The n1 and n2 range delimiters may be omitted, and thus the whole program will be listed (as in the case of --list); if you use only n1, that line will be shown; if you use the figure n1-, the listing will proceed from n1, included, to the end; if you use the figure -n2, the listing will proceed from start to n2, included; if you use the figure n1-n2, the listing will proceed from n1, included, to n2, included. If you prefer a reverse (even partial) listing, as for the LISTREVERSE and the LISTNHREVERSE commands, use option --reverse or specify n1 and n2 in inverse position. If you want the file to be saved onto the same file, use option --save. (cautions! This will overwrite the previous version, losing the special markers!)

The default listing shows the text just as it appears in the source. If you want more appeal, you can activate the -g/--grace option, which will activate three main features: 1) indenting, getting rid of the previous one, 2) spacing, separating statements from their arguments, and 3) featuring the line numbering just as it appears in the MONITOR of the DECBASIC, in five digits followed by a tab.

The indenting is three characters long by default. You can control this width by means of option -T <num>, which will set the tab to <num>, in the range 0-16; other values will be ignored and a warning will be shown.

A useful advice here is that if the file to be listed is contained into a directory that begins with a number, or if the file itself has a name that begins with a number, if you don’t specify both of the necessary numbers or if you specify only the first, followed by a dash, the path/file number will be read and used, with unpredictable results. E.g.:

```
 $ decb -L20- 2014AC.BAS
```

will fail. To avoid this, type the filename preceded by "./", or specify any second number, preceded by the dash. E.g.:

```
 $ decb -L20- ./2014AC.BAS
 $ decb -L20-99999 2014AC.BAS
```

If the listing consists of lower characters statements (deriving for instance from another BASIC), you cannot directly execute the code, because decb accepts only capital letters for statements. You can change the state of the listing by means of option --list in conjunction with option -c, to print on screen the listing, all turned to capital letters; to make this permanent, use option --save too.

As a final note, remember that if the arguments exceed the 99999 limit or are negative, they are sized to fit the limits (99999 and 0 respectively).

–N <name>
Normally, in the heading of the program, along with date and hour, the

name of the program is reported (without extension); if you want to read another name or to control this feature, you can use option -N followed by a name; if this name contains spaces or other special characters, enclose the name into quotes.

–n, ––no–header
This option tends to emulate the RUNNH command, disabling the printing of the header and the footer, making the output tighter. It can be typed also as --no-heading and --no-footer.

––no-reset
The DEC-BASIC used to reset the variables memory after a CHAIN call, so that variables instantiated in the first program were not available in the second; some other compilers, though, didn't erase this memory, so that variables could be shared between multiple programs run via a series of CHAIN statements. In order to enable the variables sharing in decb, I added this option, that can be typed also as --noreset and is available as a PRAGMA option.

–P <num>
The PAGE statement accepts a number that corresponds to the maximum page lines number set for current program (until the end of program or a new PAGE); the current limit for this number is set in decb.h to 54 (the value suitable for my printer); to use a different value greater than this you have two choices: a) change the value of DEFPAGEHEIGHT in decb.h to the value of your printer and recompile; b) use this option to pass a value that will overwrite current limit (be sure this value is acceptable, otherwise no printer will print such pages).

Before this, you've got to know which is the number of plain text pages output by your printer; to do this (if you don't know other ways), you can type:

```
 $ head -n 80 decb.c | lpr
```

and count the lines printed on the first sheet; increase the value in case the second sheet is not printed; then you can use one of the two methods explained before. (Beware: print on recycled paper!).

If you use an editor as output, you could try opening a textual file and simulate a print preview with proportional fonts, and count the lines of the first complete page.

–R [n[,f,k]], --resequence
This option tends to emulate the RESEQUENCE command, causing the listing to be renumbered from line <f>, which will receive new number <n>, and the following lines with step <k>. The inner references of GOTO, THEN, GOSUB, USING and the like will be updated as well.

You can omit all the arguments, and in this case, the default values for <f>,<n>,<k> will be used (see --resequence). If you want to specify <n> only, you can omit <f> and <k>; If you want to omit <f> or <k>, you should maintain the commas. The default value for <f> is always the first existing available line number, and the default value for <k> is

10.
––resequence is the double-dash version of -R, with default values for any of the three parameters (setting <n> equal to <f>). It can be typed also as --reseq, --renumber and --renum.

The output of the new resequenced listing will not overwrite the existing one, but it's simply printed on the terminal you use. If you want to make this change permanent, you can use option --save, which will cause the listing to overwrite the previous one. Use this last option with care. Finally, options --reverse, -T and -g/--grace have an effect on resequencing, and can also be used.

Remember, as a final note, that if you give resequencing arguments that cause numbers to go past the 99999 limit at any point, an error message is printed, and the renumbering stops.

––reverse: see option -L

–S <num>
This option tends to simulate the RUN command, which accepted <num> as argument, specifying the starting line from which to begin the program execution. If <num> is less than zero, it is set to zero; if it's greater than 99999 it is set to 99999; if the line number set as the argument is not in the listing, decb complains, and no execution is done.

––save
See option -L for its usage. I only add here that this option is very dangerous if used unconsciously, because it erases the original file and rewrites the new content. So be sure the listing output is just what you want. My advice is to test the listing without --save, to see the final effect, and use --save only when sure of the result; another helpful habit is to work on a copy of the file.

Remember finally that all the special markers (see ahead) and peculiar decb basic comments are not saved within the file if you use such option. If you need them, make a copy of the file before --save, and then copy & paste what has been deleted.

–T <num>: see option -L

INSTALLING decb

Warning: decb can be installed on UNIX systems only; on old Macs (9.x and lesser) and on Windows it isn't even guaranteed to compile; there are many differences between UNIX and not-UNIX systems for it to happen, but if you can, you're welcome.

Every care was taken, by our team, for a total compatibility among the various UNIX (MAC OS X, Linux, BSD variants, etc...)

Installing decb is very easy; access a terminal (real or emulated) and unpack the tgz package into a directory in your path; decb will unpack into a directory with the name containing version info, like

decb.1.0.beta, or something. Enter into this directory. Now, type ’make’; if you have gcc 3.x/4.x installed, it should compile without problems; if you don’t have the gcc, you probably have to correct the sources to adapt to your compiler features, but in general it’s plain C for gcc.

After compilation, install the executable. If you want to install for every user, turn to root or sudoer, and type ’make install’ (root) or ’sudo make install’ (sudoer); that’s all. If you want to install into your path only, change the destination lines in the makefile to point to a path in your directories tree and type ’make install’ (no need to become root).

After installation, just to be sure decb is up and available, type decb -h or decb -v, to see the help or the version banners. If you see them, you’re ready to go. Start recollecting your BASIC skills!

LANGUAGE STATEMENTS AND FUNCTIONS

The language understood by decb is thoroughly described in [2] and in [4]. Statements must be written in upper case letters, since decb is case sensitive; if you have only lower case letters in your listing, you can use option --list in conjunction with option -c (see previous sections).

Math and strings operators

```
+             - unary operator for positive numbers (optional)
+             - addition
+             - string concatenation
−             - unary operator for negative numbers (needed)
−             - subtraction
*             - multiplication
/             - division
^             - power (classic form)
**            - power (alternative form)
```

Relational operators

```
=             - equal
<>            - not equal
<=            - less than or equal
<             - less than
>=            - greater than or equal
>             - greater than
```

Statements

```
:             - begin a formatter image for USING
CHAIN         - transfer control to another BASIC file
CHANGE        - turn a string into its ASCII values and reverse
DATA          - store values/strings separated by commas for READ
DEF FN        - define a function, on one or multiple lines
DIMENSION/DIM - dimension an array (one or two dimensions)
END           - terminate program (required as last stetement)
FILE#         - assign sequential files channels
FILE:         - assign random access files channels
FILES         - assign files channels before program execution
FOR..TO..STEP - start a loop (also FOR..TO..BY)
GOSUB         - go to a subroutine; return after RETURN
GOTO/GO TO    - go to a specified address
IF END#       - test for end of sequential files
IF END:       - test for end of random access files
IF..THEN/GOTO - if condition is true, go to a specified address
INPUT         - input values/strings from keyboard
INPUT#        - input values/strings from sequential files
INPUT:        - input values/strings from random access files
LET           - assignment for values/strings (optional)
MARGIN        - set output margin to channels files
MAT +|−|*     - sum, subtract, multiply matrices
MAT CON       - set a matrix to all ones
```

```
MAT IDN       - set a square matrix to identity matrix
MAT INPUT     - input matrix values from keyboard
MAT INV       - invert a matrix
MAT PRINT     - print a matrix on screen
MAT READ      - input matrix values from DATA statements
MAT TRN       - transpose a matrix
MAT ZER       - set a matrix to all zeroes
NEXT I,J,K..  - terminate a FOR loop (handles multiple variables)
ON..THEN/GOTO - jump on index to a specified address in a list
PAGE/NOPAGE   - set/unset output pages size
PRINT         - print on screen
PRINT#        - print onto sequential files
PRINT:        - print onto random files
QUOTE/NOQUOTE - set/unset quote mode
RANDOMIZE     - generate a new seed for random numbers
READ          - input values/strings from DATA statements
READ#         - input values/strings from sequential files
READ:         - input values/strings from random access files
REM/REMARK/'  - comment until end of line (' may be inline)
RESTORE *|$   - reset pointer of DATA numbers/strings
RESTORE#      - set sequential files to beginning for READ
RESTORE:      - set random access files to first item
RETURN        - return from a subroutine invoked by GOSUB
SCRATCH#      - erase a sequential file and set for write
SCRATCH:      - erase a random access file
SET:          - set record pointer for random access files
STOP          - force the termination of a program
USING         - introduce a reference for a string formatter
WRITE#        - write onto sequential files
WRITE:        - write onto random access files
```

Functions

All functions ending with the dollar sign $ return strings, all the others return numbers.

```
<PA>          - write a set of void lines (page separator)
ABS           - return the absolute value
ASC           - return the ASCII code of argument
ATN           - return the arc tangent (return radians)
CHR$          - return the char matching the ASCII value argument
CLOG/LOG10    - return the common logarithm in base 10
COS           - return the cosine (argument in radians)
COT           - return the cotangent (argument in radians)
DET           - yield the matrix determinant after an inversion
EXP           - return the exponential (base e)
INSTR         - return the substring of given string
INT           - return the integer part or a number (floor)
LEFT$         - return the left substring of given string
LEN           - return the string length
LOC           - return the number of current record in file
LOF           - return the number of last record in file
LOG/LN/LOGE   - return the logarithm base e
MID$          - return the substring from a given string
```

```
  NUM           - yield the number of values input by MAT INPUT
  RIGHT$        - return the right substring of a given string
  RND           - return a random number
  SGN           - return the sign of argument
  SIN           - return the sine (argument in radians)
  SPACE$        - return a string of spaces
  SQR/SQRT      - return the square root
  STR$          - return a string from a given numeric value
  TAN           - return the tangent (argument in radians)
  TIM           - return a number in seconds since program start
  VAL           - return a value from a given string
```

Special decb markers (decb features)

```
  #             - ignore current line if it begins with it
  &             - stop loading a program if current line begins with it
                        until next line also beginning with it
  @             - stop loading a program if current line begins with it
  REM PRAGMA    - the PRAGMA statement
```

"IGNORE LINE" - If the character ’#’ is found as the first character of a line, the line will not be loaded and run at all, and any information there contained won’t be part of the BASIC execution. In practice, it is a special marker for ’Ignore Current Line’. This lets you store specific comments to the source, or disable temporarily some line, or build BASIC programs that can act as executables, if you have the first line of the program set like this:

```
 #!\path\to\decb $1
```

"SUSPEND PARSING" - If the character ’&’ is found as the first character of a line, the file loading is suspended, and following lines are read and discarded until another line beginning with ’&’ is found (and discarded too), restarting file loading with next line. In practice, it is a special marker for ’Suspend Parsing Program’. This lets you store any free form text within the source file itself, acting as temporary documentation of the source, or isolate temporarily parts of code not to be executed. Useful for comments at the beginning and in the middle of the source. The suspension may be repeated at will.

"STOP PARSING" - If the character ’@’ is found as the first character of a line, the file loading will stop, and any information thereafter contained won’t be part of the BASIC execution. In practice, it is a special marker for ’Stop Parsing Program’. This lets you store any free form text in the source file itself, acting as documentation of the source, or you can store the output of the program execution without preventing its re-execution, or any other textual information.

"SOURCE BLANKS" - A line that begins with any trailing space after the line number, is stored but not executed, just as in the original DEC-BASIC; options --list will retain such lines, making the listing more attractive.

"PRAGMA" - Let’s now discuss the PRAGMA statement; it can be enabled

after a REM o REMARK statement, like this:

```
 10 REM PRAGMA <args>
```

where <args> is a list of characters, each connected with a special function to be enabled or disabled on runtime. The PRAGMA statement must be used only once and preferably (but not obligatorily) as the very first line. This statement is a REM statement, and thus is treated as a comment by any other BASIC compiler or interpreter; its peculiar usage is to distribute source files (.BAS programs) that can be executed by decb users without the need to enable this or that option in order to fulfill the program needs (e.g. suppose the original listing expects only upper case letters, you should run the -c option from the command line, but this is not known by the user the first time he sees the file, and even if he peeps at the source he may not realize that such option is to be used; the PRAGMA statement fulfils this need automatically and makes the program independent of the command line option).

Each argument has to be followed by the letter E for ’enable’ or D for ’disable’.

Available PRAGMA arguments are (more may follow in the final 1.0 version):

```
 C  capitalization on input
 R  reset memory on CHAIN
```

Examples are (blanks are ignored):

```
 10 REM PRAGMA CE RD
 10 REMARK PRAGMA C D R E
```

Errors list

Here’s the list of codes and error messages hardwired into decb. A number of them are not implemented yet (due to current program architecture), some (surprisingly not listed in the LBMAA-A-D) were transcribed from the DEC BASIC at the Living Computer Museum, some were added to support the added features or the specific decb architecture. Where I found differences between the LCM BASIC version and the LBMAA-A-D message strings, I chose to follow the living machine...

In the following, the %s figure means that some words (strings) are substituted during the message printing at runtime, the %d figure means that a number is substituted. Moreover, every message is followed by the line number in which the error occurred (IN LINE %d) and in case of CHAINed files, the name of the file in the CHAIN where the error occurred (IN %s), except for the first file in the CHAIN.

```
 #   4 ? MEMORY ERROR (YOU MAY NOT OVERWRITE LINES OR CHANGE
 #       THEIR ORDER)
 #   5 ? MEMORY ERROR (LINE NUMBERS MAY NOT EXCEED 99999)
 #   6 ? MEMORY ERROR
 #   7 ? MEMORY ERROR (RENUM CANNOT BE PERFORMED)
```

```
#   8 ? MEMORY ERROR (NUMBER INSERTION CANNOT BE PERFORMED)
#  10 ? FILE %s NOT FOUND
#  13 ? MISSING LINE NUMBER FOLLOWING LINE %d
#  22 ? BAD DATA
#  23 ? DATA NOT IN CORRECT FORM
#  24 ? END IS NOT LAST
#  25 ? EOF
#  26 ? FAILURE ON ENTRY
#  27 ? FNEND BEFORE DEF
#  28 ? FNEND BEFORE NEXT
#  29 ? FOR WITHOUT NEXT
#  30 ? GOSUB WITHIN DEF
#  31 ? FUNCTION DEFINED TWICE
#  32 ? ILLEGAL ARGUMENT FOR ASC FUNCTION
#  33 ? ILLEGAL CHARACTER
#  34 ? ILLEGAL CONSTANT
#  35 ? ILLEGAL FORMAT
#  36 ? ILLEGAL FORMAT WHERE THE WORDS THEN OR GO TO WERE EXPECTED
#  37 ? ILLEGAL FORMULA
#  38 ? ILLEGAL INSTRUCTION
#  39 ? ILLEGAL LINE REFERENCE
#  40 ? ILLEGAL LINE REFERENCE %d
#  41 ? ILLEGAL RELATION
#  42 ? ILLEGAL VARIABLE
#  43 ? INCORRECT NUMBER OF ARGUMENTS
#  44 ? INITIAL PART OF STATEMENT NEITHER MATCHES A STATEMENT KEYWORD
#       NOR HAS A FORM LEGAL FOR AN IMPLIED LET--CHECK FOR MISSPELLING
#  45 ? MIXED STRINGS AND NUMBERS
#  46 ? NESTED DEF
#  47 ? NEXT WITHOUT FOR
#  48 ? %d IS NOT AN IMAGE
#  49 ? NO CHARACTERS IN IMAGE
#  50 ? NO DATA
#  51 ? NO END INSTRUCTION
#  52 ? NO FNEND FOR DEF FN%c
#  53 ? OUT OF ROOM
#  54 ? RETURN WITHIN DEF
#  55 ? SPECIFIED LINE IS NOT AN IMAGE
#  56 ? STRING RECORD LENGTH > 132 OR < 1
#  57 ? STRING VECTOR IS 2-DIM ARRAY
#  58 ? SYSTEM ERROR
#  59 ? TOO MANY FILES
#  60 ? UNDEFINED FUNCTION -- FN%c
#  61 ? UNDEFINED LINE NUMBER %d
#  62 ? USE VECTOR, NOT ARRAY
#  63 ? VARIABLE DIMENSIONED TWICE
#  64 ? VECTOR CANNOT BE ARRAY
#  65 ? %s WAS SEEN WHERE %s WAS EXPECTED
#  66 % ABSOLUTE VALUE RAISED TO POWER
#  67 ? ATTEMPT TO OUTPUT A NEGATIVE NUMBER TO A * OR $ FIELD
#  68 ? ATTEMPT TO OUTPUT A NUMBER TO A STRING FIELD OR A STRING
#       TO A NUMERIC FIELD
#  69 ? ATTEMPT TO READ# OR INPUT# FROM A FILE WHICH DOES NOT EXIST
#  70 ? ATTEMPT TO READ# OR INPUT# FROM A FILE WHICH IS IN WRITE#
```

```
#       OR PRINT# MODE
#  71 ? ATTEMPT TO WRITE A LINE NUMBER > 99,999
#  72 ? ATTEMPT OT WRITE# OR PRINT# TO A FILE WHICH HAS NOT BEEN
#       SCRATCH#ED
#  73 ? ATTEMPT TO WRITE# OR PRINT# FROM A FILE WHICH IS IN READ#
#       OR INPUT# MODE
#  74 ? CHR$ ARGUMENT OUT OF BOUNDS
#  75 ? DATA FILE LINE TOO LONG
#  76 ? DIMENSION ERROR
#  77 % DIVISION BY ZERO
#  78 ? EXPONENT REQUESTED FOR * OR $ FIELD
#  79 ? FILE IS NOT RANDOM ACCESS
#  80 ? FILE NEVER ESTABLISHED - REFERENCED
#  81 ? FILE NOT FOUND BY RESTORE COMMAND
#  82 ? FILE %s ON MORE THAN ONE CHANNEL
#  83 ? FILE NOT IN CORRECT FORM
#  84 ? FILE RECORD LENGTH OR TYPE DOES NOT MATCH
#  85 ? IF END ASKED FOR UNREADABLE FILE
#  86 ? ILLEGAL CHARACTER IN STRING
#  87 ? ILLEGAL CHARACTER SEEN
#  88 ? ILLEGAL FILENAME
#  89 ? ILLEGAL LINE REFERENCE IN CHAIN
#  90 ? IMPOSSIBLE VECTOR LENGTH
#  91 ? INPUT DATA NOT IN CORRECT FORM--PLEASE RETYPE
#  92 ? INSTR ARGUMENT OUT OF BOUNDS
#  93 ? LEFT$ ARGUMENT OUT OF BOUNDS
#  94 ? LINE NUMBER OUT OF BOUNDS
#  95 % LOG OF NEGATIVE NUMBER
#  96 % LOG OF ZERO
#  97 % MAGNITUDE OF SIN OR COS TOO LARGE TO BE SIGNIFICANT
#  98 ? MARGIN  OUT OF BOUNDS
#  99 ? MARGIN TOO SMALL
# 100 ? MID$ ARGUMENT OUT OF BOUNDS
# 101 ? MIXED RANDOM & SEQ. ACCESS
# 102 ? MIXED READ#/INPUT#
# 103 ? MIXED WRITE#/PRINT#
# 104 ? NEGATIVE STRING LENGTH
# 105 ? NO FIELDS IN IMAGE
# 106 ? NO ROOM FOR STRING
# 107 ? NO SUCH LINE IN RUN (NH) OR CHAIN
# 108 ? NOT ENOUGH -- ADD MORE
# 109 ? ON EVALUATED OUT OF RANGE
# 110 ? OUT OF DATA
# 111 ? OUTPUT ITEM TOO LONG FOR LINE
# 112 ? OUTPUT LINE > 132 CHARACTERS
# 113 ? OUTPUT STRING LENGTH > RECORD LENGTH
# 114 % OVERFLOW
# 115 % OVERFLOW IN EXP
# 116 ? PAGE LENGTH OUT OF BOUNDS
# 117 ? QUOTA EXCEEDED OR BLOCK NO. TOO LARGE ON OUTPUT DEVICE
# 118 ? RETURN BEFORE GOSUB
# 119 ? RIGHT$ ARGUMENT OUT OF BOUNDS
# 120 ? SET ARGUMENT OUT OF BOUNDS
# 121 % SINGULAR MATRIX INVERTED
```

```
 # 122 ? SPACE$ ARGUMENT OUT OF BOUNDS
 # 123 % SQRT OF NEGATIVE NUMBER
 # 124 ? STRING FORMULA > 132 CHARACTERS
 # 125 ? STRING RECORD LENGTH > 132 OR < 1
 # 126 ? SUBROUTINE OR FUNCTION CALLS ITSELF
 # 127 % TAN OF PI/2 OR COTAN OF ZERO
 # 128 ? TOO MANY ELEMENTS-- RETYPE LINE
 # 129 % UNDERFLOW IN EXP
 # 130 % UNDERFLOW
 # 131 ? VAL ARGUMENT NOT IN CORRECT FORM
 # 132 % ZERO TO A NEGATIVE POWER
 # 179 ? CHANNEL NUMBER IS <1 OR >9
 # 180 ? LINE TOO LONG
 # 190 ? OUT OF MEMORY WHILE DIMENSIONING AN ARRAY
 # 191 ? OUT OF MEMORY WHILE DEALING WITH VARIABLES
 # 192 ? OUT OF MEMORY WHILE DEFINING DEF FN%c FUNCTION
 # 193 ? OUT OF MEMORY WHILE STORING DATA STRINGS
 # 194 ? OUT OF MEMORY WHILE STORING LINE %d
 # 195 ? %s IS A DIRECTORY -- CANNOT EXECUTE
 # 196 ? MISSING FILE NAME
 # 197 ? CANNOT ACCESS FILE %s
 # 198 ? OUT OF MEMORY FOR NESTED GOSUB
 # 199 ? THE PRAGMA STATEMENT MUST BE USED ONCE ONLY
 # 200 ? MISSING NAME FOR OPTION -N
 # 201 % VALUE OUT OF BOUNDS FOR OPTION -%c: SETTING DEFAULT
 # 202 ? UNRECOGNIZED OPTION -%c
 # 203 ? MISSING %sARGUMENT FOR OPTION -%c
 # 204 ? RECURSION LIMIT VIOLATED
 # 205 % LINE NUMBER ALREADY REFERENCED
```

All error codes from 190 onward are specific runtime messages, generally unrelated to the BASIC language, but rather pertaining to decb inner routines. See the file errors.h for some notes about errors.
Not all the error messages have been used. For a very small number of them, I wasn’t able to detect their meaning or to conceive a way to insert them, yet. Oh my!.

A BIT OF LOGIC IN BASIC

Some DEC-BASIC coeval brothers had some operators dealing with logic (for instance the OSU Basic). This turned to be a standard after the Microsoft QuickBasic appeared on the market, but the DEC-BASIC couldn't make use of them, anyway, because they were not implemented in it. Here's the scope of this chapter: making full use of the math abilities of the DEC-BASIC implemented into decb, in the logic area, for the sake of logic itself. Remember that the proposed solutions won't be applicable in a 'real' DEC-BASIC.

Maybe you wonder why such a chapter has been put here: logic is my favorite math discipline, no surprise then.

The logic operators

The logic infix operators are commonly AND, OR, XOR, EQV, IMP, NOR, NAND; another important logic prefix operator is NOT.

Here are the logic truth tables of the prefix operators (we'll treat NOT in a separated issue). A and B are here the results of comparison tests (usually returning -1 in case of true, and 0 - zero - in case of false); for instance, for A we can use X>0, and for B we can use Y<24

A	B	AND	OR	XOR	EQV	IMP	NOR	NAND
-1	-1	-1	-1	0	-1	-1	0	0
-1	0	0	-1	-1	0	0	0	-1
0	-1	0	-1	-1	0	-1	0	-1
0	0	0	0	0	-1	-1	-1	-1

The logic math

Now, let's consider the math operators + (addition), * (multiplication) and - (subtraction). Division is not in the group, because of its weird tendency to fail if you try to divide by 0.

A	B	+	*	-
-1	-1	-2	1	0
-1	0	-1	0	-1
0	-1	-1	0	1
0	0	0	0	0

It's easy to see that the following conditions hold:

Logic operators		Logic math	Example
A AND B	==>	A*B>0	(X>0)*(Y<24)>0
A OR B	==>	A+B<0	(X>0)+(Y<24)<0
A XOR B	==>	A-B=0	(X>0)-(Y<24)=0
A EQV B	==>	A-B<>0	(X>0)-(Y<24)<>0
A IMP B	==>	A-B>=0	(X>0)-(Y<24)>=0
A NOR B	==>	A+B=0	(X>0)+(Y<24)=0

```
 A NAND B       ==>     A*B=0           (X>0)*(Y<24)=0
```

These operators cannot be used with any numerical values. Take a look at this:

```
 -1 AND 0       ==>     -1*0>0          returns 0, ok
 -1 AND -1      ==>     -1*-1>0         returns -1, ok
 -24 AND 0      ==>     -24*0>0         returns 0, ok as well, but
 14 AND -7      ==>     14*-7>0         returns 0 (should return 1)
```

The last result is to be interpreted logically, not bitwise: both 14 and -7 are true truth values, because they are not null. So I expect that 14 AND -7 is true, but it’s not. The same may be said for operators involved with sum and subtraction. A trick for using always truth values is to use any isolated value V in the form (V<>0), which is a tautology: if V is not zero, this turns to be true (-1), but if it’s zero, it will return false (0). In facts:

```
 (14<>0) AND (-7<>0) ==> -1 AND -1 ==> -1*-1>0 ==> -1, ok
```

The conclusion is: math logic is strictly bound to logical truth values.

The NOT prefix operator

While the infix operators present an easy math, the NOT operator is rather a different matter, since BASIC has no way to turn a truth value to its opposite, because it does not know what a ’truth value’ is.

Let’s begin with the NOT truth table:

```
  A    NOT
 ---------
 -1     0
  0    -1
```

Now let’s consider, using math operations, to solve our problem like before; which operation? Addition and subtraction, since multiplication and division, at most, amplify or reduce an effect. The operations will sum and subtract -1, before and after (for different exits of sign):

```
  A    A-1    A+1      1-A     1+A
 ---------------------------------
 -1     -2      0       2       0
  0     -1      1       1       1
```

Addition gives the correct result in both cases, but only **if** A is a truth value, that is if it’s -1 or 0. This method DOESN’T work with any positive true value, but it can be used with comparison operators:

```
 NOT A          ==>     A+1 or 1+A      (X>0)+1   or    1+(X>0)
```

The same previous conclusion applies here: math logic is strictly bound to logical truth values, so use the tautology previously explained to avoid using numerical values.

THE DIFFERENCES WITH THE ORIGINAL DEC.BASIC

With reference to the original DEC-BASIC (v.17F), here's a brief list of the main differences among the two systems I could find, differences that you may encounter during decb usage. I guess you can call them "known bugs"...

1. Interpretation vs. Compilation

The original DEC-BASIC (as well as its ancestor, the Dartmouth BASIC) was a compiler that, after the 'RUN' command was invoked, did the following:
- compilation of the source
- printing of messages for any error encountered
- execution in case of correct source
- in case of brutal stop by the user, two ^C were necessary (and printed, with no TIME: indication)

decb is instead an interpreter that does the following:
- read next source line
- interpret it and execute it
- the first error causes the printing of a message, and execution stops, otherwise restart the process
- in case of brutal stop by the user, one ^C suffices (though the final screen looks like the DEC-BASIC screen, with two ^C printed and no TIME: indication)

Speed, of course is an issue, but nonetheless, on my ancient 2005 emac with a ppc G4 processor, the interpreter decb is faster than the compiled original, 3 to 20 times faster! 40 years later, yes, and you bet: on a more modern computer, it should go ever better!

2. Numbers

The way numbers are stored in the PC memory influences the results of some comparison test whenever we approach some numeric limit. The DEC-20 used a 36 bit processor, while now a default power of 2 is used (32 and 64 bits, for the present time).

Take this example:

```
 10 LET A=1E27
 20 LET A=A*10
 30 LET A=A*10
 40 LET A=A*10
 50 IF A < 1E30 THEN 40
 60 PRINT A
 70 END
```

I expect that, as in line 50 A is equal to 1E30, decb does not execute the "THEN 40" part; unfortunately it does, so A=A*10 is performed again. At the end A=1.00000E+31, and this is what is printed by line 60 (the DEC-BASIC returned correctly 1.00000E+30). In effect, in line 50 A was

slightly less than 1E30, because of roundoff errors. The problem lies in the way C treats the double float numbers, and is at present an unresolvable feature.

The DEC-BASIC on its turn encountered problems in other ways, for instance not detecting integers because of its own roundoff errors; for instance, after some calculation that should return 3, a 3. is printed (3 followed by a dot) instead of 3 because the calculated value is something like 3.000000323424 and not a plain 3.

In treating results approaching zero, decb behaves better: whenever the DEC-BASIC returned something like 1.67879E-7 (to be interpreted as zero), decb returns something like 3.45657E-17 (which is much less) or even zero.

Another point: the LBMAA-A-D manual states that only numbers may be input for numeric variables with the INPUT or MATINPUT statements, with no more than 8 digits; actually, the DEC-BASIC could handle numbers at any length (tested on the LCM), and so does decb: input numbers may exceed the 8 digits rule but no formulas can be input.

The matrix inversion algorithm is another matter: the algorithm used by decb and the one used by the DEC-BASIC return the same results for a great number of cases but, for some, the DEC-BASIC algorithm seems (take care, seems) more precise than mine, giving results also for singular matrices with null determinant. My algorithm, instead, refuses to calculate an inverse in the case of singular matrices (i.e. in the case of null determinant), but this is in common with various online matrices calculators, which behave just the same. In effect, there is no inverse with null determinant; maybe the fact that the DEC-BASIC algorithm gives results when the determinant is expected to be null is because it is not null, being a roundoff error sufficient to yield a quasi-null value, and so letting the program proceed with the inversion, which of course, returns big big numbers...

Finally, the LBMAA-A-D states that two operators must not be consecutive, apart from *, ** and ^ which may be followed by + or -; decb respects this feature, but its parser is built in such a way that the unary signs + or - are tightly bound to the entity following it, so that operations like -3+-2, +3+-2, -3-+2, +3*-2 are accepted; this is a small difference with the LBMAA-A-D behavior, that extends the total abilities of this simulator.

3. Error messages

I couldn’t test all the error messages, all the error possibilities and all the serious or light mistakes in syntax and calculations. So, I relied on the DEC-20-LBMAA-A-D manual for the errors list, and tried to implement them as needed, taking care to verify a subset of them, especially when I was in doubt.

Having stated this, I must admit that in many cases error messages are not perfectly equal to the original DEC-BASIC. For instance, sometimes the DEC-BASIC printed something before emitting the error message, while

decb prints the error message immediately, or vice versa. This may depend, in some cases, on the fact that some controls are done before others, and this is in contrast with the original DEC-BASIC behavior.

But, you know, it’s not easy to test all the error possibilities. So, I tried to issue at least a plausible error message or warning, in the case of syntax errors and bad calculations, so you shouldn’t be in trouble.

If you should find yourself in trouble, write me.

4. Output and formatting

The same can be said about the output and formatting of the BASIC environment. I tried to replicate as closely as possible the numbers and strings printing, both for PRINT and MATPRINT, for matrices and vectors, for special characters and so on; sometimes, I couldn’t avoid correcting some weird things; in particular, I’m referring to the BASIC at the Living Computer Museum, where in the program banner it writes the date with a curious "25-NOV-;4", while I use the correct text "25-NOV-14" (probably the 21st century was not taken in account at all, in those times).

I expect some differences in some cases, especially when warnings are mixed to the normal output of the program, but hey! As long as you get the correct results, who really cares?

5. Logic as math

Due to the algorithm in decb (basing on the C constructs), a simple value IS a truth value, TRUE if not zero, FALSE if zero. Thus, a statement like

```
 10 IF A THEN 40
```

is perfectly legal in decb (and of course in some other BASICs too), while the DEC-BASIC returned an ILLEGAL RELATION error message. Besides, simple comparison relations are considered ’formulas’ in decb, so a line like

```
 10 PRINT (X>0)+(X=0)<0
```

is perfectly legal in decb (though its scope must be justified), while it gave an ILLEGAL FORMULA error message on the DEC-BASIC.

6. Files

Output to files mostly conforms to the original BASIC. Of course, the content of random files differs, because the behind mechanics are different (theirs and mine). But the results on the screen are the same. Some differences arise when very small MARGIN or PAGE limits are adopted, but I guess this is not a real issue.

Another issue regards IF END. decb’s IF END does not distinguish between sequential files with or without line numbers (that is in READ or INPUT

mode). This is because the C interface I built decb with admits one backup character only by means of the ungetc() C-function; so decb's IF END returns TRUE every time the stream is past EOF, without checking what this last character was. Thus, IF END: and IF END# are treated differently according to their tasks, but the mere effect is the same: test if past EOF.

Another difference is in the QUOTE statement. While the DEC-BASIC could give messages like ? OUTPUT ITEM TOO LONG FOR LINE IN LINE 30 when trying to print a string in QUOTE mode, especially with small MARGIN arguments, decb is more tolerant, and sizes the output, with or without quotes.

A final remark: the CHAIN statement in DEC cleared out all the variables content between two CHAIN calls. Moreover, I found that some programs, in other dialects, permitted the sharing of variables in a CHAIN series. Thus, in order to make these programs run with decb, I enabled this feature through option, --no-reset, which prevents the resetting of variables between two CHAIN calls.

7. GOSUB

In the LBMAA-A-D there's no mention of the limit of the GOSUB stack (the higher level of allowed nested GOSUBs); I set it to 999, considering it sufficient for any serious calculation. In any case, this limit may be increased in decb.h.

The DEC BASIC was aware of the circular calls that arise when GOSUB A calls GOSUB B that calls GOSUB A and the like. I don't know how long this chain could be in the DEC-BASIC. The decb chain in such calls may be long, up to 999 calls.

8. Recursion

In the LBMAA-A-D there's no mention about the limit of the recursion in DEF FN functions calling themselves. I set it to 511 (a higher value causes a crash on my computer), considering it sufficient for any serious calculation. In any case, this limit may be increased in decb.h.

9. The pseudo-random number generator

The DEC-BASIC had a complete and good pseudo-random numbers generator; in those times the tests for random numbers generation were beyond conception, and so this generator was not certainly tested, but nonetheless the algorithm was good enough for any BASIC user.

So is the algorithm behind the pseudo-random numbers generation in decb, which is simple and naive (you can check its sources for the technical matters), but has shown, in spite of this, a very good behaviour. I have found that this algorithm has a periodicity of 8532532 events, recurring after a variable number of events depending on the starting values. This means there are at least more than 8.5 million useful pseudo-random numbers, before the cycle repeats; I say this without considering the initial phase, before the cycle begins, that has shown a length variability

from circa 100000 (1E5) to 7000000 (7E6) numbers, and possibly more, numbers which should be also added to the count!

This said, the algorithm is not to be used for serious cryptography or important cyphering/deciphering programs, because it does not guarantee unquestionable results for a huge amount of pseudo-random numbers. If you need such a generator, build your own: it’s a very interesting intellectual challenge, believe me!

In future versions, I have planned to recover the algorithm of the DEC-BASIC, if I can, and replicate even this small aspect of the emulator. For the moment, enjoy this little gem!

TRIVIA

The vim syntax file

I am an old vim user; I couldn’t learn emacs (probably it’s a mental thing), and the other editors are simply too trivial.

In the package there’s a file called decb.vim, which may help vim/gvim users like me to write decb BASIC programs with a colored-syntax sized for the DEC-BASIC source files. Follow these instructions in order to install it:

1. Look for the file filetype.vim in the user vim directory (Programs/vimfiles/ for Windows and $HOME/.vim/ for UNIX), open it with a text editor and add the following lines to it (create this file if it doesn’t exist):

```
" decb (DEC-SYSTEM BASIC)
au BufNewFile,BufRead *.BAS     setf decb
```

2. copy decb.vim into syntax/; you may create this directory if it does not exist (Programs/vimfiles/syntax for Windows and $HOME/.vim/syntax for UNIX).

3. You’re done. Now, when you open a .BAS file with vim you will see the proper statements colored by function. Happy programming.

The decb.vim file is a work-in-progress. If you happen to find ways to better it, do it and send it back to me: I’ll include your changes and integrations in next version.

The emacs syntax file

The emacs syntax file is not ready yet. I wait for your contributions, dear emacs users...

SEE ALSO

Some documents and references played an important role during the design and coding of decb, and many of them were consulted even earlier, during the making of dib [1]. In the following, recompose broken http links (split with a dash to make them fill up the 80-columns space).

[1] dib (dib is BASIC)

My simulator of version III of Dartmouth BASIC (1966) is available here: http://digilander.it/tonibin/deer/dib. Version III was the first to include fully functional MAT features. Watch out! This program is no longer being maintained.

[2] BASIC User's Guide

This book (a.k.a. LBMAA-A-D) is a pdf printed book for the DEC-20 BASIC, version 17F, freely available at http://bitsavers.informatik.uni-stuttgart.de/www.computer.museum.uq.edu.au/pdf/DEC-20-LBMAA-A-D%20BASIC%20User27s%20Guide.pdf
The "BASIC User's Guide" is a reprint of the typewritten manual "BASIC Conversational Language Manual" (a.k.a. LBLMA-A-D), the older DEC-10 BASIC reference for the BASIC language and interface, also freely available as pdf at http://www.bitsavers.org/pdf/dec/pdp10/TOPS10_softwareNotebooks/vol06/DEC-10-LBLMA-A-D_BASIC.pdf.
A Reference card, also in pdf format, for DEC BASIC version 17B (a rapid consultative document) can as well be freely downloaded at http://bitsavers.informatik.uni-stuttgart.de/www.computer.museum.uq.edu.au/pdf/DEC-10-XBRCA-A-D%20DECsystem10%20Basic%20Language%20Reference%20Card%20(Version%2017B).pdf
These books and texts were the basis of my studies about BASIC and of my efforts in creating decb.

[3] Programming with BASIC, by Byron S. Gottfried, edition 1, 1975 (Italian version, as Programmare in BASIC, Schaum 1982)

This book describes a slightly different BASIC version than Dartmouth version V or DEC version 17B/D/F. This was my first book about programming!

[4] BASIC Sessions, by Thomas E. Kurtz, chapter XI of the book History of programming languages

The article, about the original BASIC environment, 1977 ca, was once freely available, and I downloaded it, both as a typewriter transcript (titled BASIC) and as chapter XI of the issued volume, but now ACM has retired the free availability imposing a fee to download it. Shame on it!

[5] BASIC Version 2. revision B

The CDC manual for the BASIC that ran on Cyber 70/170 and 6000 Series computers in 1974, freely available as pdf at http://bitsavers.informatik.uni-stuttgart.de/pdf/cdc/cyber/lang/basic/19980300B_BASIC_Lan-

guage_Version_2_Reference_Nov74.pdf, was also useful for the making of decb.

[6] BASIC's User Manual

The Oregon State University manual for the BASIC that ran on a Control Data 3300 computer in 1972, freely available as pdf at http://bitsavers.informatik.uni-stuttgart.de/pdf/oregonState/os3/CCM-71-08_BASIC_May72.pdf, was also useful for the making of decb.

MORE...

Dartmouth BASIC II manual

Manual for Dartmouth BASIC version II, October 1964, freely available as pdf at http://www.bitsavers.org/pdf/dartmouth/BASIC_Oct64.pdf

Dartmouth BASIC III manual

Manual for Dartmouth BASIC version III, June 1965 (rev. September 1966, printed by General Electric), freely available as a pdf file at www.bitsavers.org/pdf/ge/MarkI_Timesharing/IPC-202026_BASIC_Sep66.pdf

Dartmouth BASIC IV manual

Manual for Dartmouth BASIC version IV, January 1968, freely available as a pdf file at http://www.bitsavers.org/pdf/dartmouth/BASIC_4th_Edition_Jan68.pdf

John Kemeny

News about the main inventor of BASIC, prof. John Kemeny, may be found at http://cis-alumni.org/JKemeney.html.

Thomas Kurtz

News about the co-inventor of BASIC and strict Kemeny's friend and collegue, Thomas Kurtz, may be found at http://cis-alumni.org/TKurtz.html.

Mean Matrices

This article by Valentin Albillo on ill-conditioned matrices appears on Vol. 24 of the HPCC Club Magazine, and was once available as a pdf file at http://membres.lycos.fr/albillo/calc/pdf/DatafileVA014.pdf but it's no more, and I cannot find it elsewhere. Sorry.

Zero to the Zero Power

by Su, Francis E., et al., in Mudd Math Fun Facts, http://www.math.hmc.edu/funfacts/ffiles/10005.3-5.shtml where you can find an explanation to the fact that zero to the zero power is 1. Check also http://mathforum.org/dr.math/faq/faq.0.to.0.power.html where it's stated: "Consensus has recently been built around setting the value of

0^0 = 1".

The Living Computer Museum (LCM)

This wonderful organization offers a free access to the mainframes in their possessions; in particular, a DEC-10 system with version 17F of BASIC is available. You can reach them at http://www.livingcomputermuseum.org/.
I used extensively their machine to test some specific behaviors, but beware: being driven by a Windows server, it’s often off-line, due to maintenance or crashes. Be patient, write to Rick and wait... it’ll soon be back!

BUGS

Of course, bugs are everywhere. If you find bugs, mail me to <ing dot antonio dot maschio at gmail dot com>; include the Operating System identification with version, the gcc version you used to compile, the program you were running, the extract of the execution (if you can copy from the terminal) in a .txt file, and a brief description of what you were about to do and what you got instead. English or Italian, please. A mixture will do, of course! ;-)

HISTORY

decb appeared in 2014 for the first time. Written by Antonio Maschio, engineer in Montebelluna (TV), Italy - near Venice - with active help by I.J., programmer in Bristol, England.