DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
MAYNARD, MASSACHUSETTS/TWinoaks 7-8822/TWX MAYN 816

December 9, 1963

Miss Jane Levy
Room 26-260
Massachusetts Institute of Technology
Massachusetts Avenue
Cambridge 39, Massachusetts

Re: DECUS No. 31

Dear Jane:

A copy of one review is enclosed and the following is quoted:

"The program we finally worked with was the automatic multiply/divide portion of the tape labeled "Digital-1-18a-S-MB" dated 8/21/63. Paul McKiernan of our engineering staff did most of the bug finding. His difficulties arose both from program defects and write-up errors. He made no attempt to correct the program tape since he was able to avoid the difficulties by slightly modifying specified procedures and adding a few rules. These have been assembled in a revision of the write-up which I am enclosing along with a copy of the working tape.

EDC also exhibits some unspecified features. An error type-out, "sce", appears to indicate full buffer space but allows all previously loaded definitions to be used without error. Also, it was found that sense switch 6 controls a punching option though it is not clear how to use this effectively."

We are looking forward to your contribution to DECUSCOPE and the DECUS Program Library.

Sincerely,

Elsa Newman (Mrs.)
DECUS Secretary

EN:ajc
Enclosures: Copy of Program Review

DECUS

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
MAYNARD, MASSACHUSETTS/TWinoaks 7-8822/TWX MAYN 816

see - Sense switch controls Punch.

November 27, 1963

Professor J. B. Dennis
Electrical Engineering Department
Massachusetts Institute of Technology
Room 26-258
Cambridge 39, Massachusetts

Dear Professor Dennis:

You were missed at the DECUS Meeting on November 18 and 19! It was a good meeting but somewhat saddened by the absence of our founding colleague, Ben Gurley. The quality of the presentations were outstanding for such a small group. We have initiated work on the 1963 PROCEEDINGS and will send you copies of them.

When the 1962 PROCEEDINGS were published, I was very frugal and now find I have a few extra copies. Please distribute these among students or wherever you feel they may do some good.

With reference to the DECUS Program Library, I have two "reviews" by users indicating possible errors in the M.I.T. Expensive Desk Calculator Program. When I called M.I.T. this morning I spoke with Mr. Ralph Butler about obtaining new tapes. He kindly offered to run two. I am hopeful this meets with your approval and that you may wish to have him check the faulty tape I enclose. As you know, DECUS has been concerned with procedures for "review" of programs prior to their certification. Any other contribution you may wish to make to the DECUS Program Library will be appreciated. The format of the write-up is forwarded for your convenience.

Looking forward to hearing from you again.

Sincerely,

Elsa Newman

Elsa Newman (Mrs.)
DECUS Secretary

Enclosures: 1962 PROCEEDINGS (3)
DECUS No. 31 (write-up and tape)
Write-up Format

DECUS PROGRAM REVIEW

PROGRAM

Expensive Desk Calculator LPC Digital 1-18-5-1 RIM
LPC Digital 1-18-5-2 RIM Symbol

REVIEWER

M. Hill (for J. Mott-Smith, AFCRL, PLP)

Digital tape 1-18-5-1 RIM must be used in conjunction with digital tape 1-18-5-2 RIM (its symbol tape). Before the tapes are read in, the MUS and DIS switches inside the cabinet must be thrown up.

In checking the arithmetic computations on page 3 of the write-up, Expression - 6 X 3/7 yields an answer of -3. To get a 3 decimal answer, the expression must be - 6 X 3.000/7. This point is brought out in the following paragraph on decimal digits. Likewise, the expression 100-3 x 4/9 + 6 must contain specific decimal notation in the numerator (either 3.000 or 4.000) to get 104.667. Otherwise it prints 105. Using no decimal significance, 4 x 9/7 x 11, yields 55. Using 3 decimal significance, 4 x 9/7 x 11 yields 56.573 but 4 decimals produce 56.5719 and 6 decimals produce 56.571427.

In the paragraph concerning decimal digits in division, (e) states that the expression 1/2 + .0000 yields 1. The expression 1/2 + .0000 yields 1.0000, rounded to the 0 places of the 1/2, as does expression (a).

The paragraph concerning significance (page 4) states that typing NS, where N is some integer less than 40, causes computations to retain results accurate to N decimal digits. However, typing 39 S prior to typing 4 X 9/7 X 11 causes type out of 19 significant decimal figures followed by 19 zeroes.

1.42

On page 6 in the section on implied multiplication the expression (t - 1/(n+2), n - 1/(n+2), n + t), t) should be written (t - (1/(n + 2), n - 1/(n + 2), n + t), t).

The iteration procedure described on page 7 printed out 0 if no significance level is specified. If a decimal level is chosen, it will repeat the formula until the value of 1/n becomes 0, unless sense switch 3 is on. Then the typing is suppressed.

Using the form of an exponent as described on page 8 - E <SIGN><DIGITS>, adds numbers when the sign is + and subtracts when the sign is -. 2E <+> <2.0> equals 4.0, 3 E <+> <2.0> = 5.0, 2E <-> <2> = 0, 5.00E <-> <2> = 3.00.

The field size control operates as explained. If the field size is 4, typing in .99998 prints out 1.0000, typing 1.00000 prints 1.0000, but typing 1.01000 prints as 1.010.

In computing and printing the results of a table of values for the function $y = x^2 + 3x + 4$ for values of x from 0 to 100 in steps of 1, the write-up states that the results will be typed as a single column of alternate values of x and y (with ss3 on):

x_1

y_1

x_2

y_2

.

.

.

Using the iteration brackets, as suggested < x tab xxx + 3 x + 4 tab ((x+1), x - 101)> prints out values of the function y alternating with the values of x in steps of 1 with x equal to (x+1) times (x-101) starting with x = 0 and ending with x = 100. Hitting the tab after < x causes a print out of 0 and carriage return. Again hitting tab after typing x x x + 3 x + 4 causes a print out of 4 and carriage return. Thereafter - 10

8

- 18

14

- 24

22

.

.

.

.

Recalculating the same function, $y = x^2 + 3x + 4$, for values of x from 0 to 100 in steps of 1, but using < x = UCTAB, again produces alternating values of x and y to be printed out. However, for x it prints the correct value of x followed by (x + 1) times (x - 101). The form is: 0

4

1 - 10

8

2 - 18

14

.

.

.

A better arrangement of values of x and y typed in tabular form with each numeric value of x and the resulting value of y on the same line with proper spacing between the print-outs of the column, can be produced by shifting from upper case to lower case and back to upper case following the equal sign typing and before the UCTAB. That is,

<x=UCTAB is typed as ⇑<⇓x⇑=⇓⇑→.

This prints out

0	4
1	8
2	14
.	.
.	.
.	.

A great deal of time and many trials were completed to use the macro, POLY, to produce the above table. The write-up is not clear in its use of the middle dot. On page 7 of the write-up it states "in order to define an abbreviation or "MACRO", type the desired name followed by a middle dot (.). EDC will shift into red and enter the MACRO DEFINE MODE. In this mode no computations are done." Further on in the same paragraph, it states, "Middle dot is the character used to leave the MACRO DEFINE MODE." However, nowhere in the write-up does it say to hit middle dot, shifting to Macro Define Mode, and immediately hit middle dot in the same space, leaving the Macro Define Mode to make write-up of POLY work. Typing middle dot twice in the same space after typing poly, then space bar before typing 0 is necessary to produce the print out of the numerical values of x and y in tabular form of 2 columns.

The section describing macros as functions operates as described provided sense switch 3 is up and the center dot is typed twice in the same space followed by a comma.

DECUS No. 31

NUMBER: Digital-8-S

NAME: EXPENSIVE DESK CALCULATOR

AUTHOR: Robert A. Wagner - MIT

DATE: January 2, 1963

SPECS: Uses all of memory
RIM

NEEDED: Typewriter

ABSTRACT: EDC provides for performing arithmetic operations on numbers typed either on or off line, and printing results. Decimal numbers (integers, decimal fractions or integer-fraction combinations) are acceptable; all indicated by ordinary decimal point conventions. EDC allows the internal storage of "variable" registers. The names of such registers, when used in the same contexts as typed numbers, automatically cause their current contents to be used in the calculation, as if the contents had just been typed in. EDC stores arbitrary character strings for later use as input to EDC, and for testing the sign of partial results.

EDC provides means for performing arithmetic operations on numbers typed either on or off line, and printing results. Decimal numbers consisting of integers, decimal fractions or integer-fraction combinations are acceptable, all indicated by ordinary decimal point conventions. The output of EDC is essentially the same format. In addition, EDC allows the internal storage of often used quantities and of partial results, in named "variable" registers. The name of such registers, when used in the same contexts as typed in numbers, automatically cause their current contents to be used in the calculation, as if the contents had just been typed in. In addition, means for storing arbitrary character strings for later use as input to EDC, and for testing the sign of partial results are all provided.

SIMPLE COMPUTATIONS:

1. Numbers: A number is a string of digits of any length ≤39, which may or may not include a decimal point. If a decimal point is present, it may appear anywhere within the digit string, or at either end of it. A number which does not contain a decimal point is treated as an integer.

2. Operators: An operator is one of the special characters + | <space> | - | / | ×. (Note: the symbol "|" means "or".) The meaning of each of these operators is as follows:

Operator	Meaning
+ or <space>	add
-	subtract
×	multiply
/	divide

These operators can be used to cause EDC to perform arithmetic operations on numbers.

3. Accumulator: EDC, like many desk calculators, contains an internal "working" register where results are accumulated. The register may be cleared to zero by typing <carriage return>. Alternatively, its contents may be typed out before it is cleared. This is accomplished by typing <tab>.

At this point sufficient concepts have been introduced to allow the user to perform arithmetic operations on numbers he types in and to obtain correct results.

An expression in EDC consists of several numbers separated by

operators. Each operator uses as one of its two arguments the number typed immediately after it. (If no number is typed, a zero is assumed). Since not all the operators listed above associate, it is necessary to define the order in which operations are performed when more than one operator appears in an expression. Within any expression all multiplications and divisions are performed before any additions and subtractions. Except for this rule, all operations take place from left to right.

Examples of valid expressions:

Expression	Meaning	Equals
-1	-1	-1
2.x3-4	(2x3)-4	2
-6x3/7	-((6x3)/7)	-2.571
100-3x4/9+6	100-((3x4)/9)+6	104.667
4x9/7x11	((4x9)/7)x11	56.573

DECIMAL DIGITS

The number of digits to the right of the decimal point in a number defines the number of decimal digits in the number. The number of decimal digits in the result of any computation is always the larger of:

(a) the number of decimal digits retained in the expression at the time the computation is performed, and

(b) the number of decimal digits in the argument of the operator specifying the computation.

This is particularly important in the case of division. The division operation rounds the quotient produced to the number of decimal places specified in the above rule. Thus,

(a)	1/2	yields	1,
(b)	1.0/2	"	.5,
(c)	1/2.000	"	.500,
(d)	.0000+1/2	"	.5000,
(e)	1/2+.0000	"	1.

In example (a) both the 1 and 2 are specified to zero decimal places. The answer, 1, is really .5 correctly rounded to zero decimal places.

In examples (b) and (c) one of the factors was specified to more than zero decimal places. The answer is computed accurate to a number of decimal places equal to the larger of the number of decimal places specified in either factor. In example (d) the .0000 specifies that the expression is hereafter to retain 4 decimal places. Hence the division is accurate to 4 places. Example (e) illustrates what appears to be an inconsistency. However, at the time the division is performed, the numbers 1 and 2 are accurate to only 0 places. When the .0000 is typed, the result of the division is all that remains of the original 1 and 2. Thus the quotient cannot be re-evaluated and remains rounded to zero places when the .0000 is added in. Moral: Type a number which specifies the number of decimal digits retained in a division before attempting the division.

SIGNIFICANCE:

In example (d) above, the number of decimal places to be retained in future computations was specified by typing ".0000" as the first number in the expression. An exactly equivalent operation which allows the user to conveniently specify the number of decimal places to be retained in all succeeding computations is provided. Typing NS, where N is some integer less than 40, causes all succeeding computations to retain results accurate to N decimal digits.

PARENTHESES:

Any expression may be enclosed in parentheses. As in algebra, the value of expressions enclosed in parentheses is computed before operations outside the parentheses are performed. Actually, in EDC typing (EXPRESSION) is equivalent to typing a number equal in value to the value of EXPRESSION.

VARIABLES:

EDC provides means for storing intermediate results internally and using the stored results in later computations. This is accomplished by means of a notation called "variables". In form, a variable consists of a string of letters of arbitrary length. (Actually only the last 3 letters are significant.) A quantity may be placed in a variable (and the variable "defined") by typing:

NUMBER, NAME, where NUMBER is a number or its equivalent, and NAME is a string of letters. This causes the value of NUMBER to be stored in the variable NAME. The number may appear as a part of an expression. More of the expression may follow the variable definition. In particular, another variable definition may store the same number in still another variable. Note: The storing is not accomplished until some character other than a letter is typed following

the first letter in the name. If the name is mistyped, it may be deleted by typing an overbar ($^-$) before any non-letter is typed.

Once a particular name has been used as the name in a variable definition, it may be used as an equivalent to a number. The value of this type of number equivalent is the contents of the variable at the time it occurs in an expression. If it appears again as a name in a variable definition, the new number replaces the old contents of the variable.

To summarize:

A number is (1) a string of digits with or without a single decimal point, or (2) (expression), or (3) variable. Thus, the following are numbers:

1
1.
(1)
(-1)
(3x5+1/4-6x3/2x9)

If a is a variable, then

a
(a)
((a))
(3xa+5)

are numbers.

IMPLIED MULTIPLICATION:

Since in EDC three different types of numbers exist, it is possible to assign a meaning to the juxtaposition of two numbers not both of type 1 or type 3. This meaning has been chosen to be "multiply", just as if an "x" had been present between the two numbers. For example, if a is a variable, then

3a	3a4	3(4(3))	a3a
3a+5	(3)4(3)	a(4(3))	a(3)a

are all expressions, and, for example,

3a means 3xa.

Note that if you wished to multiply 3 by 4, you would have to write "3(4)," or (3)4, rather than "34," since this, of course, is the decimal integer thirty-four. Similarly, to compute a squared,

assuming a is a variable, you must write axa, or a(a), or a1a, rather than aa, since this last notation represents a new variable whose name is "aa".

Examples of variable definitions:

(1+3),a
(3x4),abc

Note:

(-1),a puts -1 in a
-1,a puts +1 in a, since the variable definition operates on the last number typed before the comma.

Once a is defined as a variable, (a+1),a is legal, causing the contents of register a to be increased by 1.

The following is also legal and is a number:

(t-1/(n+2),n-1/(n+2),n+t),t)

(Assuming, of course, that n and t had been previously defined.)

In order, the above expression

(1) adds the old value of t into the number being computed
(2) increments n by 2
(3) inverts (takes the reciprocal of) this new value of n
(4) again increments n by 2
(5) subtracts the inverse of this new value of n from the first computed inverse
(6) adds to this difference the old value of t
(7) stores the new value in t
(8) subtracts this new value of t from the old value saved previously.

ITERATION:

A simple means is provided for allowing EDC to repeat a procedure several times and stop automatically. This feature is provided through the brackets ≤ and ≥. If S is an arbitrary string of characters (which may include bracket pairs <...>), ending in a

number, n, then

<S>

will cause the string S to be re-interpreted each time n is computed and found to be negative. NOTE: Zero is positive in the arithmetic scheme used by EDC.

For example,

```
0,t
(-1),n
<(1/(n+2),n-(1/(n+2),n),k+t),t
(-k)>
```

computes pi/4 by the formula

pi/4= 1-1/3+1/5-1/7. . . . . .

The result is left in register t. The computation terminates when the value of 1/n is computed to be zero. This will, of course, be dependent on how many digits the significance level is set to.

MACROS:

It is often convenient to have some means of remembering some sequences of operation. In EDC, provision is made for abbreviating arbitrary strings of characters by completely independent names. These names, when expanded, supply the original string of characters automatically from memory to the rest of the processor. Thus often-used sections of the major computation need be typed only once. Whenever the particular computation is needed, it can be performed by merely stating the abbreviation chosen to designate this particular computation.

In order to define an abbreviation or "MACRO", type the desired name followed by a middle dot (·). EDC will shift into red and enter the MACRO DEFINE MODE. In this mode no computations are done. Instead each character typed is entered into storage. All characters except middle dot, overbar and backspace may be so entered for later interpretation when the macro is expanded. The three characters which cannot be entered into storage each have special functions in this mode. Middle dot is the character used to leave the MACRO DEFINE MODE. The other two characters are provided to facilitate correcting long or involved MACRO's. Backspace is to be used to delete characters, one by one, from the stored character string. Each typed backspace causes the last remaining character in this macro's storage area to be deleted. Overbar has a function analogous to the "start read" key on a Flexowriter. If the particular

abbreviation chosen for the macro has been previously defined, the new definition will completely replace the old. However, while the new definition is in progress, an overbar will cause the first character in the old definition's string to be typed, deleted from the old definition's string, and added to the end of the new definition's string. If the end of the old definition is reached, an overbar will be typed but will not enter the new definition's string. If sense switch 2 is on after a character is entered in the new buffer, EDC acts as if overbars were given until the switch is turned off. This allows rapid copying of the remaining portion of an old definition.

Once a MACRO has been defined, it may be expanded by mentioning its name at any time, followed by some character which is not a letter. This character, the "break" character, will not be interpreted immediately. Instead, it will appear as the character following the last character in the MACRO expansion. Normally, when a MACRO is being expanded, the characters in the expansion are typed out on-line. This may be suppressed by turning on sense switch 3. In fact, whenever EDC is in the "automatic" mode, either as the result of iterations or macro expansions, sense-switch 3 on will suppress type-out of the characters being spilled.

OTHER OPTIONS:

Paper tapes prepared on the standard FIO-DEC flexowriter may be used as input to EDC in place of the on-line typewriter. When sense-switch 1 is ON and some character is typed (to cause EDC to leave its typewriter listen loop), EDC will read characters from a flexo tape in the reader until a stop-code is reached. These characters will be acted on exactly as if they came from the typewriter keyboard. When a stop-code is reached, EDC returns control to the listen loop, allowing the user to turn SS1 off or to type some character.

A number may be immediately followed by an exponent, indicating that the number represented is the number typed, multiplied by 10 (decimal) raised to the indicated power. The form of an exponent is

E<SIGN><DIGITS>

where <SIGN> is +, SPACE, -, or is not present

and <DIGITS> is a string of digits, representing a decimal integer.

At least one digit should appear in the string <DIGITS> if the resultant number is to be followed with a sign.

Any number, or number equivalent may have an exponent supplied to scale its values by integral powers of ten. However, it should

be noted that the value of the exponent is subtracted from the number of decimal places in the number typed immediately before the E. The effect of E is thus merely to move the decimal point.

Note: <DIGITS> must be an integer and will be taken modulo 2^{16}.

FIELD SIZE CONTROL

Some control is provided over the total number of digits printed before an E by EDC. This is accomplished by a field size character, F, in the context <NUMBER>F. Hence, number must be an integer <40. The new print field size becomes effective when an F is encountered and remains effective until a new F is typed.

The output number is correctly rounded to the digits printed, and the position of the decimal point is correct as printed, modified by the signed number following any output E, just as on input. Note: This control is approximate because rounding of a number like .99998 to 4 printed digits causes an extra digit to be introduced. Thus, the above number will print as 1.0000. In addition, any number which prints as 1, followed by no digits other than zero, will have an extra digit printed. For example, if the current field size is 4, the number 1.00000 will print as 1.0000 although the number 1.01000 prints as 1.010.

One possible use for macros is in computing and printing several results in a specified order. For example, suppose that a table of values for the function

$$y = x^2 + 3x + 4$$

is to be computed for values of x ranging from 0 to 100 in steps of 1. This particular problem can easily be solved by using the iteration brackets. One might try:

```
0,x
<x tab xxx+3x+4 tab ((x+1),x-101)>
```

and EDC will cooperate by typing a single column of alternate values of x and y (with SS3 on):

x_1
y_1
x_2
y_2
.
.
.

It is obvious that means for listing values in position other than at the far left edge of the paper would be desirable. the operators =, UCTAB and UCCAR are provided to assist in this formal control. = is an operator similar to TAB in its effect -- that is, it causes a numerical typeout. However,

1) it types the "number" typed immediately before the =, rather than the expression, as does TAB
2) no carriage return is typed following the digits
3) the "accumulator" is not cleared by the =. UCTAB and UCCAR (tab or carriage return typed in upper case) are ignored by the processor, but type a tab or carriage return regardless of the position of SS3. Using these new operators, the problem can be re-solved as follows:

```
0,x
<x=UCTAB
x×x+3x+4 tab  ((x+1),x-101)>
```

Now, although a table of values in acceptable form has been produced, the first value of x and that of y are found intermixed with portions of the user's typing. To help sort them out and to preserve the completed program for further use, the entire character string typed by the user could be defined to be a macro called, say, POLY:

```
poly◦0,x
<x=UCTAB CARR
x×x+3x+4 tab  ((x+1),x-101)>
```

Among other advantages, defining the string as a macro allows use of the macro editing sense switches to correct typographical mistakes.

MACROS AS FUNCTIONS:

A properly defined macro can operate in EDC as if it were a number (of type variable for purposes of implied multiplication). In addition, it is possible for a macro to take one argument from the expression in which it is used. Thus, for example, it is possible to define a macro which replaces the last number typed with that number's absolute value. The general technique is to write a macro whose first operation is that of storing the last number in a unique variable. For example, the definition

```
name; x<+(-x),x>
```

allows

```
(a-b)name
```

to compute the absolute value of the number (a-b), leaving the result in x. This occurs because the string of characters represented by the abbreviation name is:

,x<+(-x),x>

Hence, typing (a-b) name is equivalent to typing

(a-b),x<+(-x),x>

The iteration <+(-x),x> changes the sign of the contents of variable x repeatedly until the sign is positive.

Note: the plus sign following the < is provided to avoid the useless computation of -x×x which would result wherever x was originally positive. Addition in EDC is somewhat faster than multiplication and should be the preferred operation. One difficulty with this macro is that it fails to supply the result in a convenient form for further computations. Two operators have been provided which simplify the operation:

N, and C

Both are "deletion" operators and may be so used even outside macros.

N zeros the last number typed.

C zeros the current expression only back to the last unpaired open parenthesis. (The rest of the current expression is untouched.)

Using these operators, the macro "name" can be rewritten as follows:

name;xN(<+(-x),x>Cx)

Using this definition of "name", let us follow EDC's computation of

(1)name

The character string seen by the processor is, in effect:

(1),xN(<+(-x),x>Cx)

After the N is interpreted, the value of x is 1. and the accumulator contains zero, giving the effect that no number was typed since the operator which preceded the (1). In effect, then, the number (1) has been deleted from the string seen by the processor, although the value of this number is safely preserved in x. Now, the computation inside the parentheses is performed, and when the C is interpreted, the sum is deleted from the accumulator without

affecting the value of any part of the expression which was typed before the first. Since the "answer" is contained in X, X is now added into the expression, and the parenthesis count is reduced to its value before the macro was spilled.

Using the definition of "name", either

10(a-b)name

or

((a-b)name)10

computes 10 times the absolute value of (a-b).

Similarly, (a-b) name/3 computes one-third the absolute value of (a-b).

Given the following two macro definitions, "sqr" becomes a square root function:

aa;xxN(<+(-xx),xx>C-xx).

sqr;xN(x,y<+((y+x/y).5),z+(y-z,y)aa>Cy).

Now the number 9sqr has the same value (to the number of figures specified by the last S operation) as does 3.