S. Howry, Jan. 1969

The Desk Calculator on the Beam Switchyard Computer

1. INTRODUCTION

This note describes an additional feature of the beam switchyard control computer (BSYC) system - a desk calculator which allows the operator to make short on-line computations without disturbing any of the control activities of the computer. The overall control system language and functions described in references 1,2,3, will not be given here. Since this will also serve as a "how to use it" manual for users, there are appendices with example calculations and library functions. Also there is a section on polish strings because these constructs make up the "syntax" of the calculator, and should be understood by the user if he is to compute to full advantage.

The BSYC system provides for simultaneous use of the desk calculator by a person at the Beam Switchyard Control area and one at the Spectrometer building, communicating via the computer/computer link between these two points. However, a program which ships raw teletype characters between the link and a teletype must be integrated into the spectrometer computer system.

2. POLISH STRINGS

All expressions typed into the calculator must be in polish string form. This is because of the severe memory restrictions placed on the program. In principle an infix-to-polish Translator could be interfaced to the calculator. On the other hand, once the polish is understood it is not so bad and does offer advantages of simplicity in certain cases, e.g. vector valued function evaluation (see Complex Package in Appendix C below).

In this description an identifier is a string of characters delimited by one or more spaces. Identifiers are normally typed into the desk calculator via ASR-35 teletype on-line to the BSY computer. The desk calculator program interprets these one-by-one deciding which ones are arguments (operands) and which are functions (operators). In a polish string a function symbol follows all of its

required arguments. Common arithmetic expressions such as:

1 + 2

in polish notation are:

1 2 +

and more complicated ones such as

(1 + 2)/(3 + 4)

or

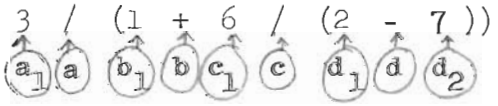
3/(1 + 6/(2 - 7))

become respectively:

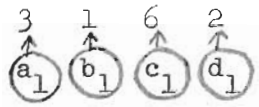
1 2 + 3 4 + /

3 1 6 2 7 - / + /

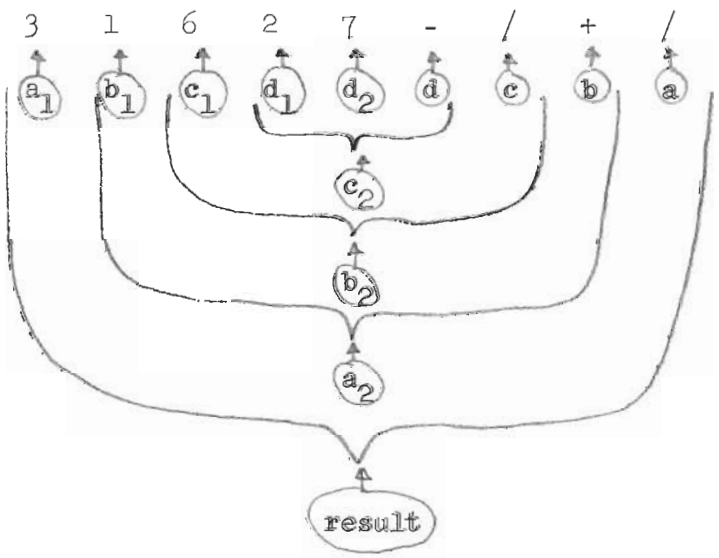
Notice that no parentheses are required. For this reason the polish strings are sometimes referred to as a "parenthesis free" notation. Most commercially available desk calculators effectively operate with this notation. One always loads both numbers into the calculator (e.g. one is in its "accumulator" and the other in the keyboard), then hits the function key (e.g. the '+' button). A prime example of such a calculator is the Monroe EPIC 2000 printer.[4] For a detailed description of polish string notation, the reader is referred to the operating instruction manual for this calculator. The last example above will now be converted to Polish to give the user an idea of how this is done. To make this example clear the various functions and arguments will be labeled as indicated below:



Scanning from left to right in the usual way, when the first function ('/') appears in the algebraic expression there is only one argument. Hence it is deferred until the second argument appears. Similarly the '+' second '/', and '-' must be deferred while awaiting their second arguments so the operands are placed on the argument stack:



When the 7 appears, the '-' has two arguments and so this function may be evaluated. The expression "2 7 -" creates c_2 the second argument for the '/' function labeled c. (c) . This single argument then replaces the two arguments d_1 , d_2 on the stack. Now since c_1 , c_2 are available, they are replaced by the results b_2 of applying function (c) to them. Continuing through to the final result in this manner:



3. IDENTIFIER TYPES

Each identifier is recognized by the desk calculator as one of five different types. These types are listed below, together with a description of the corresponding action taken by the computer.

(i) Numeric Constants

Numeric constants are identified by the first character. If it is a digit, a sign, or decimal, the computer creates a floating point number from the characters of the identifier. By using the E operator, much like in FORTRAN, one can get a power-of-ten factor with the number. However, unlike FORTRAN, a space must separate the mantissa from the power-of-ten. Thus

12.345

1.2345 E1

1234.5 E-2

+12.345

12.34XYZ5

12.345E

12.345CM

are all representations of the same floating point number. The last 3 illustrate the fact that non-numeric characters are ignored once a leading numeric is encountered. The computer uses only the first six decimal digits of the identifier. For this reason it is better to write "6.12 E-5" rather than ".0000612". Numeric constants are considered as operands and hence placed on the argument stack.

(ii) Basic Functions

Certain identifiers are automatically interpreted as functions. In general one or more arguments have been entered onto the stack and the function replaces them with one or more results. The difference Δ between number of arguments stacked after and before is a useful number and will be given for all the basic functions. A summary of all the functions is in Appendix B.

+ -replaces the top (i.e. last) two arguments on the stack with their sum: (next-to-top arg) + (top arg). Hence $\Delta = -1$.

Error cases:

In the case when the top argument is not a number, the function is a 'no operation' with $\Delta = 0$. In case when the top argument is a number but the other is not, the function removes the number but does nothing else ($\Delta = -1$).

- -replaces the top two arguments on the stack with their difference: (next-to-top arg) - (top arg). Hence $\Delta = -1$.

Error cases: same as '+' above

* -replaces the top two arguments on the stack with their product. Hence $\Delta = -1$.

Error cases: same as '+' above

/ -replaces the top two arguments on the stack with their quotient: (next-to-top arg)/(top arg). Hence $\Delta = -1$

Error cases: Same as '+' above for illegal arguments. Division by zero yields zero.

EXP -replaces the top argument on the stack with its antilogarithm: exp (top arg). Hence $\Delta = 0$

Error cases: If the top argument is not a number the function is a 'no operation' with $\Delta = 0$. If the argument is too big, the result is zero.

LN -replaces the top argument on the stack with its natural logarithm: $\ln_e$ (top arg) Hence $\Delta = 0$

Error cases: If the top argument is not a number the function is a 'no operation' with $\Delta = 0$. If the argument is less than or equal to zero, the result is zero.

P -prints the value of the top argument and removes it from the stack. Hence $\Delta = -1$. If it is an alphanumeric constant the first 4 characters of it are printed, left adjusted. Otherwise it is a numeric value, which results in an 8 character printout, 6 decimal digits, a decimal point, and sign (if negative)

Error cases: If the number is larger than 2^{40} in magnitude, the computer prints 'TOO BIG'. If less than 2^{-40} in magnitude the computer prints a single zero digit.

The last basic functions are involved with storage of arguments in a small random access memory. In using the desk calculator, it is often convenient to be able to permute the arguments in the stack or to duplicate an argument.

For example

6*7/7.1 + 3/(1 - 6*7/7.1)

would be computed by

6 7 * 7.1 / 3 1 6 7 * 7.1 / - / +

But notice that the result of the underlined part is computed twice. The storage functions allow such results to be stored away and recalled as desired.

S -this function stores the top argument on the stack in a 'scratch pad' memory, and removes it from the stack. Hence $\Delta = -1$.

Error cases - none

L - this function loads a value from 'scratch pad' memory onto the top of the stack. Hence $\Delta = 1$

Error cases - none

S and L provide communication between the argument stack and one fixed memory cell. There are up to 99 such locations available using the functions S1, L1 --- etc.

S1
S2
⋮
S99

-analogous store functions providing 99 additional scratch pad locations

L1
I2
⋮
L99

-analogous load functions from the 99 additional scratch pad locations.

S.
S1.
S2.
⋮
S99.

-These functions store the top argument on the stack into scratch pad memory exactly as do functions

S
S1
•
•
S99

described above. However the top argument is not removed from the stack. Hence for these functions $\Delta = 0$.

With these functions, recomputation in the above example is avoided by computing it with the sequence:

6 7 * 7.1 /, S. 3 1 L - / +

(iii) User defined macro names

The most useful feature of the desk calculator is its capability to define macros - sequences defining more complicated functions -- which can be recalled and executed by entering a single identifier.

How to define and use macros is best described by a specific example such as that of the previous section. The recurring portion of

6*7/7.1 + 3/(1 - 6*7/7.1)

namely,

6*7/7.1

may be defined by a macro:

```
TERM    [ 6  7  *  7.1  / ]
```

Then the example is evaluated by:

```
TERM  3  1  TERM  -  /  +
```

Another example: suppose f(E,I) = E*2 + 8/(E*I + 4.125)
is a commonly used function of the arguments E and I. Then a function definition:

```
F [ S S1 L1 L1 * 8 L1 L * 4.125 + / + ]
```

is defined, along with:

```
FP  [ F  P ]
```

To compute and print this function for E = 15.01, and I going from .030 to .035 in steps of .001, the user types:

```
15.01   .030   FP   { Computer answers here }   (user hits carriage return
                                                  for formatting)
15.01   .031   FP              "
15.01   .032   FP              "
15.01   .033   FP              "
15.01   .034   FP              "
15.01   .035   FP              "
```

Thus the identifiers F and FP are functions, equivalent to the built-in ones of section (ii) above. A list of useful macro definitions is in Appendix C.

The general form of a macro definition is:

⟨alpha constant type identifier⟩ [⟨polish string⟩]

-after the opening bracket '[' is encountered the alpha constant type identifier (see section (v) below) becomes a macro name, and is associated with the polish string which follows before the terminating closing bracket ']'. As a side effect of the definition the stack is cleared of <u>all</u> arguments.

Warning - an opening bracket must be matched with a closing bracket; also brackets within brackets are not allowed.

The general form of macro execution is:

⟨macro name⟩ - causes the polish string associated with the macro name to be executed

Δ = (number of stack arguments after executing macro) - (number of stack arguments immediately preceding execution of macro)

Use of scratch pad memory within macros is valid only within the definition of that macro. Thus if:

A = [S L P]

B = [S L 1. + A L]

The scratch pad memory location used by macro B is <u>not</u> the same as that used by macro A. A new scratch pad array is allocated by the desk calculator whenever a macro execution begins, and is discarded when that execution is complete. This has the advantage of making the macros 'relocatable' as regards to scratch pad memory. On the other hand communication of variables from one macro function to another must be via the argument stack rather than scratch pad memory, since there are no global variables in the latter.

Efficiency note: Use of lower scratch pad locations such as S Sl, is recommended rather than high ones like S99. This is because the lengths of the allocated scratch pad arrays are determined by the highest location referenced. Thus in the above example macro B's scratch pad array is of length 1, but if B were redefined as:

B = [S9 L9 1. + A L9]

it would be of length 10. These locations are protected during the execution of macro A.

(iv) BSY Control VAriables

The desk calculator can refer to certain magnet parameters that are available to the BSY Computer System. The complete list of such parameter names is given in Appendix B. When such a name is encountered by the calculator, the argument taken is the numeric floating point constant equal to the most recent raw input value (millivolts or flipcoil units) that the system has for that parameter. Example - suppose the DVM for the Q10 channel reads 112.2 millivolts at the instant that

.5 Q10 *

is typed. The result is the number 56.100 .

(v) Alpha Constants

An identifier is assumed to be an alpha constant if it is not any of the special types described above i.e. if it is not:

a numeric constant

a basic function

a user defined macro name

a BSY control variable name

Alpha constants may be printed with the P operator and so are useful to format results. Only the first four characters of alpha constants are retained. Thus:

ABCDEF

ABCD••//*-+XYZ

ABCD1234

are all the same alpha constant and (if it is not a user macro) it will print as:

ABCD

A useful alpha constant is the single character created by the 'backspace' key on the teletype, denoted by ∂ . By convention the two character alpha constant produces a carriage return when printed. Example:

if: $f_1(x) = x/(1+x)$

$f_2(x) = 1/(1+x)$

then:

```
F1   [ S L 1 L + / ]
F2   [ S 1 1 L + / ]
F12  [ S L F1= P F1 P ∂ P
          L F2= P F2 P ∂∂ P ]
```

provides the macros that facilitate creation of table below. Then to execute:

user types		computer replies		
1.	F12	F1= .500000	F2= .500000	2
1.1	F12	F1= .513809	F2= .476191	2
1.2	F12			

4. BSYC SYSTEM COMMANDS WHICH INTERACT WITH THE DESK CALCULATOR: (see reference 1)

Since there is no occasion to use the semicolon identifier in the desk calculator, there is little chance that a BSY control function will be inadvertently activated by desk calculator work. However, certain BSY instructions do affect the desk

calculator. These are listed below:

; -a single semicolon character (delimited on both sides by spaces) has the effect of clearing the argument, or operand stack, but not the scratch pad memory or user-defined macros.

CARDS; - as mentioned in the "BSY computer control language description"[1] this instruction transfers the input source to the card reader. After such an instruction, user-defined macros punched on cards may be read into the computer without disturbing the real-time BSY system. The functions in Appendix C form the start of a card function library to which users are encouraged to contribute.

✓ - this symbol (7,8- punch on a card) transfers input source to typewriter

CLEAR DC ; -this instruction clears out the desk calculator - macros, scratch memory, and stack, all. It is sometimes used to bail the user out of trouble when he tries a strange combination of desk calculator/BSY control language words. Also it is the only means at his disposal to redefine a given macro name.

5. EXECUTION ERROR FLAGS AND CONDITIONS

There are several error flags which may occur in the course of evaluating a function:

STK UNFL -a function expects an argument on the stack and the stack is empty.

Example - starting with an empty stack the sequence:

1 2 + *

will produce this error, since the function '*' has only one argument. Recovery consists of merely re-entering the correct sequence.

MEM OVFL
STK OVFL } -the calculator attempts to store arguments into scratch pad memory or onto the stack and they are full. The total memory

space set aside for the macros, scratch pad memory and argument stack is 256 locations. The only restriction is that these three components taken together cannot exceed this size. Recovery may be achieved by removing arguments from the stack (by entering a function such as 'S'). If this fails one must resort to 'CLEAR DC ; '

RECURSION OVFL -a macro has called a macro which in turn called a macro, ... etc. to a depth of 10. This will always happen, for example, if the macro calls itself as in:

```
A  [  A  ]      (defines A)
A               (executes A)
```

Note that it may be desirable to create such a function, as the next sequence shows :

```
POWR  [  S  S1  L  P  ə  P  L1  L1  L  *  POWR  ]
```

Then if x represents some numeric constant:

```
x  1  POWR
```

will print the answer

$$1 \quad x \quad x^2 \quad x^3 \quad --- \quad x^9$$

before stopping on RECURSION OVERFLOW. Since the stack is not destroyed by this halt, it contains $1 \quad x^{10}$. Hence by entering again

```
POWR
```

one generates:

$$x^{10}, \quad x^{11}, \quad x^{12} \quad ---- \quad x^{19}$$

before it halts again.

Appendix A: Exercises in polish notation

For convenience, assume that A,B,C,D,E,X are defined as numeric constants:

A	[0.4]	D	[1.1]
B	[2]	E	[-2.753 E-3]
C	[3.14159]	X	[-3]

Algebraic expression	Corresponding polish string
$\frac{A}{B + C}$	A B C + /
$\frac{A + B}{C}$	A B + C /
$\ln \left(\frac{A + B}{C}\right)$	A B + C / LN
$\frac{AB + C}{C - DE}$	A B * C + C D E * - /
$3X^4 + 4X^3 + 5X^2 + 6X + 7$	
or:	
$(3X + 4)X + 5)X + 6)X + 7$	X 3 * 4 + X * 5 + X * 6 + X * 7 +
X^4	X X X X * * * or: X X * X * X * or: X X S. L *

Appendix B: List of reserved symbols:

1.) Basic Functions

In the lists below, (top) denotes the value of quantity at the top of the stack and (top-1) denotes the value of the quantity immediately "below" the top of the stack. Δ denotes the net gain of stack operands caused by executing the corresponding function.

Basic Function	Δ	Description
+	-1	replace (top),(top-1) by (top)+(top-1)
-	-1	" " " " (top)-(top-1)
*	-1	" " " " (top)*(top-1)
/	-1	" " " " (top)/(top-1)
LN	0	replace (top) by $\log_e$ (top)
EXP	0	replace (top) by exp(top)
P	-1	print, and remove from stack, the value (top)
S. S1. --- S99.	each 0	- store in scratch pad memory the value (top): leave it also in the stack
S S1 --- S99	each -1	- store in scratch pad memory, and remove from stack, the value (top)
L L1 --- L99	each +1	- load from scratch pad memory a new value onto the stack. The new value is now the top of the stack.

2.) BSY Control Variables

PM1A	Q12	B36
PM2A	Q13	EFA1
PM3A	Q14	EFB
PM4A	Q20	ECA
PM5A	Q21	Q30
PMV	ADUM	Q31
Q10	B1T	Q32
Q11	B29	Q33
		Q34

3.) Special symbols and functions

[

]

ə (backspace key)

əə (2 hits of backspace key)

CLEAR DC ; - clears desk calculator

CARDS ; - transfers input to card reader

✓ - transfers input to typewriter

; - BSYC systems command terminator

Appendix C Macro library

The following user defined macros are included in a library deck to be found below the 925 card reader in the DAB.

1) Common functions

macro name	definition	comment
-U	[S 0 L -]	unary minus: replace (top) with -(top)
**	[S LN L * EXP]	exponentiation: (top-1)**(top)
SQRT	[.5 **]	
SIN	[S L L * S2 L2 -5040 / 1 120 / + L2 * 1.-6 / + L2 * 1 + L *]	6 figure accuracy for |(top)| ≤ π/2
COS	[S L L * S2 L2 -720 / 1 24 / + L2 * 1.-2 / + L2 * 1 +]	much worse for larger values.
TAN	[S L SIN L COS /]	
LOG10	[LN 2.30259 /]	
SINH	[S L EXP S 1 L / L - .5 *]	
COSH	[S L EXP S 1 L / L + *5 *]	

2) Complex package - defines +C -C *C /C the four basic complex arithmetic operators, along with the unary (complex) minus -CU and unary reciprocal operator /CU. Also a complex print operator PC is defined.

```
+C    [ S S1 S2 L1 + L L2 + ]
 CU       [ S -1. * L -1. * ]
-C    [ -CU +C ]
*C    [ S S1 S2 S3 L3 L1 * L2 L * - L3 L * L2 L1 * + ]
/CU     [ -1 * S S1 L1 L L1 L -1. * *C S3 S3 L1 L3 / L L3 / ]
/C*   [ /CU *C ]
PC    [ S P L P J P ]
```

Examples of use of complex package:

user types						computer responds
2	3j	4	5j	+C	PC	{ 6.00000 8.00000j }
2	3j	4	5j	-C	PC	{ -2.00000 -2.00000j }
2	3j	4	5j	*C	PC	{ -7.00000 22.0000j }
2	3j	4	5j	/C	PC	{ .560975 .048780j }
4	5j	2	3j	/C	PC	{ 1.76922 -.153846j }
2	3j	4	5j	/C		
4	5j	2	3j	/C	*C PC	{ .999992 - .000001j }

The last two lines verify that

$$\left(Z1/Z2\right) * \left(Z2/Z1\right) = 1$$

References

1. S. Howry, R. Scholl, E. Seppi, M. Hu, D. Neet "The SLAC Beam Switchyard Control Computer" IEEE Transactions on Nuclear Science, June 1967.

2. S. Howry SLAC PUB-248 "A concise on-line control system" October 1966

3. S. Howry SLAC CGTM 10 BSY Control Computer System Language, Aug. 1966

4. The Monroe EPIC 2000 Printing Calculator, Users Manual, Monroe Calculator Corp.