SLAC COMPUTATION GROUP
Stanford, California

CGTM No. 181
November 1976

```
****************
*              *
*    ABACUS    *
*              *
****************
```

ABACUS

An Interactive Expression Evaluator

Leonard Shustek
Computation Research Group
Stanford Linear Accelerator Center
P.O. Box 4349
Stanford, California 94305

```
****************
*              *
*    ABACUS    *
*              *
****************
```

TABLE OF CONTENTS

Introduction

ABACUS is a simple interactive expression evaluator available as a program under ORVYL or as a multiple-user subsystem under MILTEN. ABACUS was originally written at SLAC in the Fall of 1974. Some of its features are:

FORTRAN-like expression evaluation

Standard mathematical functions

User-defined variables

Pseudo-variables and functions for terminal I/O

IF-THEN-ELSE expressions

Looping expressions (FOR, UNTIL, WHILE)

Hexadecimal input and output, when requested

User-defined functions

WYLBUR-like modification of commands, expressions, and functions

File storage of user-defined functions

To run ABACUS when ORVYL is up, simply type "CALL ABACUS". (You, or your LOGON file, must have already executed a "SET ORV" command.) To temporarily return to WYLBUR, type "wylbur"; to then return to ABACUS, type "abacus". To terminate ABACUS, type "exit".

To start a MILTEN subsystem version of ABACUS if ORVYL is not up and ABACUS is not already running, type "EXEC FROM WYL.CG.LJS.LIB#ABACUS ON CAT". This will start an ABACUS job which will allow any number of people to sign on simultaneously. When the job has started, type "abacus" as a command to WYLBUR to enter ABACUS. If you have started the ABACUS job, it would considerate not to cancel it if other people are using it.

The prompt for ABACUS is a colon (:) to distinguish it from the normal WYLBUR prompt (?). The response to a colon prompt may be an expression or a command. What follows is a brief explanation-by-example of the expressions; the full syntax is included at the end.

(1) SIMPLE EXPRESSIONS

FORTRAN-like arithmetic expressions may be entered directly. More then one may be entered on a single line.

```
: 2+2
4

: 1+3*sin(.34)
2.0004613

: exp(50.5)/2   atan(.34)
4.2740671E+21  0.32773851
```

The following elements may be used in such expressions:

a) Decimal constants: 2   2.2   -2.2e-4

b) Hex constants (constants with leading zeroes and no decimal point):   01c43e   0a   012   0413a469b0
If the number of digits after the zero is less than 9, it is interpreted as a right-adjusted fixed-point integer. If there are 9 or more digits after the zero, it is interpreted as a left-adjusted floating-point number.

c) Functions (most FORTRAN functions are available):
    sin(0.46)   sqrt(abs(log(.5)-1))
(You can also define functions; see section 6)

d) Variables (8 characters or less)
Two variables are predefined: PI and E

e) Operators. The following list is in increasing order of operator strength:

```
+ -    binary (2-operand) addition and subtraction
* /    binary multiplication and division
**     exponentiation
+ -    unary (prefix) plus and minus
!      factorial  n!=gamma(n+1)
```

f) Parentheses may be used wherever needed to override the operator precedence rules.
  2*(a+b)   (n+1)!

All expressions are evaluated using double-precision floating point arithmetic. Operations which involve only integers and integer functions are computed to more precision than integer arithmetic would allow (at least 53 bits) and no low-bit errors will occur. Results are printed in what ABACUS deems to be a 'reasonable' format depending on the value; in particular, integers are printed as such.

(2) ASSIGNMENT EXPRESSIONS

Variables are created by assigning them a value.

```
: x=sin(pi/3)
0.8660254
```

Since assignments are expressions, they may be used as part of larger expressions.

```
: sqrt(y=exp(3.4))
5.4739474
```

Undefined variables are prompted for when their value is needed. Any expression can be used to define them.

```
: log2(bignum)
UNDEFINED VAR
BIGNUM: 2**8
8
```

(3) PSEUDO-VARIABLES and PSEUDO-FUNCTIONS

INPUT (abbreviated INP), OUTPUT (abbreviated OUT), and OUTNL are functions that can be used to input and output values to the terminal. These are useful primarily in looping statements and functions.

The argument of INPUT should be one or more variable names, whose values will be requested at the terminal. The argument of OUTPUT or OUTNL should be one or more variables or expressions, whose values will be written to the terminal. OUTPUT writes the values but does not necessarily go to the next line; OUTNL forces a new line to begin after the values are written. OUTNL is particularly useful in conjunction with FOR-loops for printing tables. OUTNL can also be used without arguments to force a new line.

The value of these pseudo-functions is the value of the last argument.

```
: input(a)**2                     : 1+output(a,3*4)
a: 3                              a=3  12  13
9
```

INPUT, OUTPUT, and OUTNL can also be used as pseudo-variables, that is, without an argument list. Whenever INPUT is used in an expression, a new value will be requested from the terminal; whenever OUTPUT or OUTNL is assigned a value, that value will be written to the terminal.

There is a special variable named AC whose value is always the last number written to the terminal.

(4) LOOPING EXPRESSIONS

(A) FOR Loops

An "Algol-style" FOR-TO-BY loop can be used. The body can be a single expression or multiple expressions enclosed in DO...END.

```
: s=0; for i=1 to 10 s=s+i
0  55

: for x=0 to 10 by .5  do y=x**2  s=s+y end
772.5
: y
100
```

The initial value and the increment are assumed to be 1 if omitted. If the terminating value is omitted, the loop continues until the attention key is pressed or an error is encountered.

The "value" of these expressions (the result printed automatically) is the last expression evaluated in the body. Often, however, other results will be stored in variables or printed from within the loop by the OUTPUT pseudo-function or pseudo-variable.

```
: for i outnl(i,exp(i))
I=1  2.7182818
I=2  7.3890561
I=3  20.085537
I=4  54.598150
I=...                         (ATTN pressed)
ABORTED
```

(B) LOOP...REPEAT

A sequence of statements enclosed between LOOP and REPEAT will be executed repeatedly until one of the following occurs:

1. An embedded WHILE logical-expression is false
2. An embedded UNTIL logical-expression is true
3. The ATTN key is pressed
4. An error occurs

```
: loop out=sin(inp) repeat
INPUT: 1
0.84147098
INPUT: pi/2
1
INPUT: ***      (ATTN pressed)
ABORTED

: loop inp(x); while x>0; out(sqrt(x)) repeat
X: 100
10
X: 10
3.1622777
X: 0
0         (returns last expression evaluated and stops)
```

The logical-expression used for WHILE, UNTIL, and (see next section) IF-THEN-ELSE can be any combination of arithmetic expressions with the relational operators LT, LE, GT, GE, NE, EQ (or < <= > >= ¬=) and the logical operators NOT, AND, OR. (Note that '=' can often not be used as an abbreviation for 'eq' because it conflicts with the use of '=' as the assignment operator within expressions).

The semicolons used to separate expressions in the examples are optional.

(5) IF-THEN-ELSE EXPRESSIONS

The IF-THEN-ELSE construction is a conditional expression; its value is the THEN expression if the condition is true, and the ELSE expression otherwise. If the ELSE is omitted, the value is 0 if the condition is false.

```
  : loop output = if inp(x)<0 then sin(x) else 1-cos(x) repeat
  X: 1
  0.45969769
  X: -1
  -0.84147098
  X: ***         (ATTN pressed)
```

The THEN and ELSE expressions are not limited to simple arithmetic; they can be any of the complex expression types available, or a list of expressions enclosed in DO ... END.

```
  : FOR X IF SIN(X) LT .5 THEN DO OUTNL FOR Y TO 5 OUT(X*Y) END
3  6  9  12  15
4  8  12  16  20
5  10  15...        (ATTN pressed)
   ABORTED
```

(6) DEFINING FUNCTIONS

Functions may be defined (or re-defined) with the DEFINE command. Those functions may then be used wherever a built-in function may be used.

```
   : define hyp(a,b)=sqrt(a**2+b**2)

   : define sumsq(x) = sum=sum+x**2 ; outnl(x**2)

   : def f(x) = if x le 1 then 1 else x*f(x-1)

   : def table = for i outnl(i,exp(i))
```

Functions may be defined to have zero or more formal arguments, which are local to the body of the function. Functions may be called recursively. Functions defined with the same name as a built-in functions will supercede the latter until erased. The body of a function may be a single expression or multiple expressions separated by semicolons; the maximum length is about 150 characters. The value of the function is the last expression evaluated.

The single-line restriction for functions is occasionally inconvenient, and a facility for multi-line functions may eventually be added. In the meantime, complicated functions may be constructed by "chaining", that is, by having functions call other functions as the last expression.

(6A) MODIFYING FUNCTIONS

The "modify" command allows modifications to be made to an existing function without entirely retyping it. The function is displayed and can be modified in the same way as the WYLBUR modify command for lines of the active file.

```
      : mod hyp
               HYP(A,B)  =  SQRT(A**2+B**2)
      ALTERS?                        r-
               HYP(A,B)  =  SQRT(A**2-B**2)
      ALTERS?
      :
```

(6B) SAVING AND RESTORING FUNCTIONS

Functions which have been defined can be saved in a file for later use. In the ORVYL version of ABACUS, the PUT and GET commands transfer functions between ABACUS internal storage and ORVYL files. In the MILTEN subsystem version of ABACUS, the SAVE and USE commands transfer functions between ABACUS internal storage and WYLBUR datasets or libraries. Note that, unlike the WYLBUR or ORVYL commands of the same name, the active file does not participate in the transfer. In both cases the syntax of the commands is much like those of ORVYL or WYLBUR.

ABACUS under ORVYL examples:

```
    : put chebyshev
    : get tri.diagonal user mhw gro cg
```

ABACUS under MILTEN subsystem examples:

```
    : save #fcts
    : save laplace on scr001
    : use fourier on wyl003 user ohs gro th nclist
```

Notes:

- The operand "nolist" on the "use" or "get" command prevents the functions from being listed as they are read.

- The initial default volume is "CAT", which may be changed by the "set volume" command (q.v.).

- If you attempt to save a dataset, member, or file which already exists, it will prompt with "OK TO REPLACE?". You may anticipate the question and add 'rep' to the command.

- Sequential datasets are always created in WYLBUR edit format. Library members are saved in the appropriate format for the library. ORVYL files are (currently) saved in PRINT format.

- In addition to function definitions, files that are loaded may contain any valid ABACUS commands (except another "use", "save", "get", or "put" command). You may create such files from WYLBUR. (ORVYL files should be saved in PRINT format.)

(7) BUILT-IN FUNCTIONS

1. Functions of one argument

A. Trigonometric

(Angles are assumed to be in radians unless changed by a "set degrees" or "set radians" command.)

SIN COS TAN ASIN ACOS ATAN

B. Hyperbolic

SINH COSH TANH

C. Logarithmic

EXP LOG LOG10 LOG2

D. Miscellaneous

SQRT

ERF ERFC

GAMMA LGAMMA

E. Truncation, Conversion

ABS (absolute value)

INT (truncation towards zero, INT(1.5)=1, INT(-1.5)=-1)

FLOOR

CEILING (can be abbreviated CEIL)

DEG (convert to degrees from radians)

RAD (convert to radians from degrees)

2. Functions of two arguments

COMB(N,K)	The number of combinations of N things taken K at a time. COMB(N,K)=N!/(K!*(N-K)!)
PERM(N,K)	The number of permutations of N things taken K at a time. PERM(N,K)=N!/K!
MOD(A,B)	The remainder of A divided by B. (Extended to generalized residue for non-integral arguments.) Examples: mod(10,3) is 1, mod(3.4,1.5) is 0.4

3. Miscellaneous functions

DUMP(address,length)	This returns the value of the 'length' bytes in memory beginning at 'address'. 'length' must be 1, 2, 3, or 4, and is assumed to be 4 if omitted.

(8) COMMANDS

Commands may be entered at any time. If the prompt is for a variable, the same prompt will be re-issued after the command has been executed. Most keywords can be abbreviated with the first three or more letters.

8A. "SET" Commands

set degrees	Change to degrees for trig functions. (Default is radians)
set radians	Use radians for trig functions.
set digits n	Display numbers with n significant digits. ($2 \leq n \leq 16$) (Default is 8)
set hexadecimal	Display results in hex. This doesn't affect input; hex numbers are still entered with a leading zero. (Default is decimal output.)
set decimal	Return to decimal cutput.
set fuzz xxx	Set the relative tolerance for floating point comparisons to xxx. The initial fuzz is 1.4E-14.
set mode retry	In "retry" mode, the previous command or expression will be listed to allow modification when an error occurs, or when a null line (just a carriage return) is entered (see Note 3).
set mode noretry	In "noretry" mode, the previous command or expression will be listed for modification only in response to "@<ATTN>". See note 3. The default mode is "noretry".
set volume vvvvvv	Change the default volume for "use" and "save" commands to vvvvvv. The default volume is "CAT".

8B. "SHOW" Commands

show digits	Display current output precision.
show radians show degrees show trig	Display current trig mode. ("degrees" or "radians")
show vars show variables	Display all variables and their values.
show fcts show functions	Display all functions, both built-in and user-defined.
show ufcts show ufunctions	Display only user-defined functions.
show mode	Show whether "mode retry" or "mode noretry" is in effect.
show volume	Display the current default volume.
show fuzz	Display the relative tolerance used for floating-point comparisons.
show hex show decimal	Display the current output mode. ("hexadecimal" or "decimal" output)

8C. Other Commands

modify xxx	Display function xxx and allow it to be changed. The modifications are entered exactly as if the Wylbur "modify" command had been given for a line of the active file. See Note 3.

erase var1 var2...	Erase the listed variables. (This is useful if the variable table has overflowed. The current limit is 25 variables.)
erase fct1 fct2...	Erase the listed user-defined functions. (This is useful if you are getting "no more room" or "stack overflow" error messages.)
erase *vars	Erase all variables.
erase *fcts	Erase all user-defined functions.
erase *	Erase all variables and user-defined functions.
suggest text...	Send a suggestion. (About ABACUS, presumably. Comments, complaints, etc. are most welcome.)
help	Give a brief explanation of some ABACUS features.
to xxx text...	(Same as regular TO command)
wylbur	Return to WYLBUR.
wylbur xxx	(ORVYL version only) Execute xxx as a WYLBUR or ORVYL or MILTEN command, then return to ABACUS.
@xxx	(ORVYL version only) An abbreviation for "WYLBUR xxx".
milten	Return to MILTEN.
milten xxx	Execute xxx as a milten command.
logoff logon	(Same as MILTEN logoff/logon. WYLBUR will NOT prompt you with OK TO CLEAR?)
exit	(ORVYL version only) Terminate ABACUS and return to ORVYL immediately. All function definitions and defined variable will disappear.
use ...	See section 6B.
save ...	" " "
get ...	" " "
put ...	" " "

Notes about commands:

1. Any unrecognized SHOW or SET commands are sent to MILTEN. In the ORVYL version, they are also sent to WYLBUR and ORVYL.

2. Most long keywords can be abbreviated by their first three or more letters.

3. Wylbur-like modifications (in response to the prompt "ALTERS?") may be used in the following circumstances:

   A) After a "modify <fctname>" command

   B) After appending "@<ATTN>" to the end of an input line

   C) By entering "@<ATTN>" as an input line (the previous command or expression will be listed for modification)

   D) By entering a null line (just a carriage return) in "retry" mode.

   E) After an error has occured in "retry" mode.

   All the Wylbur editing characters are allowed (r,d,i,l,n, and b) except "¢<ATTN>" and "$<ATTN>".

   Prompts for alteration will continue until either:

   A) A null response (just a carriage return) in which case the function will be re-defined or the command re-executed.
   B) <ATTN> as the only input. The function will be unchanged or the command will not be re-executed.

(9) DETAILED EXPRESSION SYNTAX

```
Notation:   [cptional item]      { choose this }
                                 {   or this   }
                                 {   or this   }

      The superscript o means the item may be repeated.
```

General Arithmetic Expression

```
              { <ID> = <AEXPR>                                     }
              {                                                    }
              { for <VAR> [=<AEXPR>] [to <AEXPR>] [by <AEXPR>]     }
              {                                                    }
              {                                            <AEXPR> }
              {                                                    }
              {          r              ┐o                         }
              {          |until <LEXPR>|                           }
              {          |              |                          }
              { loop     |while <LEXPR>|  repeat                   }
<AEXPR> ::=   {          |              |                          }
              {          |    <AEXPR>   |                          }
              {          L              ┘                          }
              {                                                    }
              { do [<AEXPR>]o end                                  }
              {                                                    }
              { if <LEXPR> then <AEXPR> [else <AEXPR>]             }
              {                                                    }
              { <AEXPS>                                            }
```

Simple Arithmetic Expression

```
                               r           ┐o
                               |  +<ATERM> |
     <AEXPS> ::= <ATERM>       |           |
                               |  -<ATERM> |
                               L           ┘
```

Arithmetic Term

```
                                 r             ┐o
                                 | *<AFACTOR>  |
     <ATERM> ::= <AFACTOR>       |             |
                                 | /<AFACTOR>  |
                                 L             ┘
```

Arithmetic Factor

```
     <AFACTOR> ::= <APRIM> [**<APRIM>]°
```

Arithmetic Primary

```
                                { <VAR>    }
     <APRIM> ::= [+] [-] { <CONST>  }  [!]
                                { (<AEXPR>) }
```

```
<VAR> ::= [          <ID>           }    (Variable, 0-arg. function)
          {      <ID>(<AEXPR>)      }    (1-argument function call)
          { <ID>(<AEXPR>,<AEXPR>) }    (2-argument function call)
```

```
<ID> ::= A letter followed by 0-7 letters or digits
         with no imbedded blanks.

<CONST> ::= A constant - fixed point, floating point, or
            hexadecimal - with no embedded blanks.
```

Logical Expression

```
<LEXPR> ::= <LTERM> [or <LTERM>]°

<LTERM> ::= <LFACTOR> [and <LFACTOR>]°

                                   {lt|< }
                                   {le|<=}
<LFACTOR> ::= [not] <AEXPR> {gt|> } <AEXPR>
              [ ¬ ]              {ge|>=}
                                   {eq   }
                                   {ne|¬=}
```