File Cy:

BRL R 1749

# BRL



REPORT NO. 1749

# SYMBOLANG - A SLIP EXTENSION FOR ALGEBRAIC MANIPULATION

M. A. Hirschberg

November 1974



USA BALLISTIC RESEARCH LABORATORIES
ABERDEEN PROVING GROUND, MARYLAND

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

| REPORT DOCUMENTATION PAGE | | READ INSTRUCTIONS BEFORE COMPLETING FORM |
|---|---|---|
| 1. REPORT NUMBER<br>BRL REPORT NO. 1749 | 2. GOVT ACCESSION NO. | 3. RECIPIENT'S CATALOG NUMBER |
| 4. TITLE *(and Subtitle)*<br>SYMBOLANG - A SLIP EXTENSION FOR ALGEBRAIC MANIPULATION | | 5. TYPE OF REPORT & PERIOD COVERED<br>6. PERFORMING ORG. REPORT NUMBER |
| 7. AUTHOR(s)<br>M. A. Hirschberg | | 8. CONTRACT OR GRANT NUMBER(s) |
| 9. PERFORMING ORGANIZATION NAME AND ADDRESS<br>USA Ballistic Research Laboratories<br>Aberdeen Proving Ground, Maryland 21005 | | 10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS<br>Program Element 62618A<br>Project 1T662618AH80 |
| 11. CONTROLLING OFFICE NAME AND ADDRESS<br>US Army Materiel Command<br>5001 Eisenhower Avenue<br>Alexandria, VA 22304 | | 12. REPORT DATE<br>NOVEMBER 1974<br>13. NUMBER OF PAGES<br>29 |
| 14. MONITORING AGENCY NAME & ADDRESS*(if different from Controlling Office)* | | 15. SECURITY CLASS. *(of this report)*<br>UNCLASSIFIED<br>15a. DECLASSIFICATION/DOWNGRADING SCHEDULE |

16. DISTRIBUTION STATEMENT *(of this Report)*

17. DISTRIBUTION STATEMENT *(of the abstract entered in Block 20, if different from Report)*

18. SUPPLEMENTARY NOTES

19. KEY WORDS *(Continue on reverse side if necessary and identify by block number)*

| | |
|---|---|
| Algebraic Manipulation | Formula Manipulation |
| Symbol Manipulation | Data Manipulation |
| List Processing | Formula Translation |
| List Manipulation | Symbolic Processing |
| | String Processing |

20. ABSTRACT *(Continue on reverse side if necessary and identify by block number)*

SYMBOLANG, originally developed by A. Lapidus, M. Goldstein, S. Hoffberg, and expanded by H. Bernstein, S. Greenspan, A. Magnus, and others is a collection of FORTRAN-callable subroutines which can perform arithmetic operations (addition, subtraction, multiplication, division, etc.), substitutions, evaluations, and differentiations on expressions represented as SLIP lists. SYMBOLANG consists of seventy-two subprograms and is operational on the BRLESC II computer.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE *(When Data Entered)*

TABLE OF CONTENTS

# I. INTRODUCTION

## A. Background

SYMBOLANG was written by Lapidus,[1] Goldstein and Hoffberg between 1965 and 1967 and extensively expanded by Bernstein,[2] Greenspan, Magnus and others in 1969. It is a collection of FORTRAN-callable subroutines which can perform arithmetic operations (addition, multiplication, subtraction, division, etc.), substitutions, evaluations, and differentiations on expressions represented as SLIP lists.[3,4,5] It allows the performance of such operations as multiplying the expression $(1 + x)$ by the expression $(1 - x)$ to obtain the expression $(1 - x^2)$ rather than some numeric value. SYMBOLANG is one of several formula manipulation[6] or algebraic manipulation systems in existence.

SYMBOLANG has been used extensively. It has been employed in generating coefficients for Taylor series, coefficients for the solution of partial differential equations with boundary values, expansion of determinants, and differentiation of complicated expressions. In addition to the many subroutines provided in SYMBOLANG, the user has the ability to define new functions at will and incorporate them into the system.

SYMBOLANG is an extension of SLIP, which SYMBOLANG uses as a list processing system. The virtue of using a list processing system for embedding a symbol manipulator is that the space needed to treat

---

[1] Lapidus, A., and Goldstein, M. Some Experiments in Algebraic Manipulation by Computer. Communications of the Association for Computing Machinery, 1965, 8 501-508.

[2] Findler, N. V., Pfaltz, J. L., and Bernstein, H. J., Four High-Level Extensions of Fortran IV: SLIP, AMPPL-II, TREETRAN, SYMBOLANG. New York, Spartan Books, 1972, 305-387.

[3] Weizenbaum, J., Symmetric List Processor. Communications of the Association for Computing Machinery, 1963, 6, 524-544.

[4] Hirschberg, M. A. SLIP for the BRLESC II Computer, BRL R 1731, July 1974

[5] Findler, N. V., Pfaltz, J. L., and Bernstein, H. J., Four High-Level Extensions of Fortran IV: SLIP, AMPPL-II, TREETRAN, SYMBOLANG. New York, Spartan Books, 1972, 1-82.

[6] Sammet, J. E., Survey of Formula Manipulation. Communications of the Association for Computing Machinery, 1966, 9, 555-569.

expressions expands and shrinks over a large range. A list processing language allows the allocation of space needed; unneeded space can be returned to the system. In SYMBOLANG, it is the user's responsibility to return unneeded space to the system.

SYMBOLANG consists of seventy two subroutines and is operational on the BRLESC II computer. The SYMBOLANG-SLIP system is stored on the disc and is accessed by use of the following statement:

```
* COMPILE DISC, SYLANG, ALL
```

SLIP is stored on the disc and is accessed by use of the following statement:

```
* COMPILE, DISC, SLIP, ALL
```

SLIP is written primarily in FORTRAN and allows the programmer to create lists, insert and delete items, and scan items on a list. In SYMBOLANG a SLIP list may be thought of as parenthesized group of quantities. Since a SLIP list may be placed on a SLIP list, one must take care never to place a list on itself.

We will assume the reader has a knowledge of FORTRAN and a working knowledge of SLIP in the sequel of the text.

Expressions are represented as SLIP lists in the following manner (using the Backus-Naur form):

$$< \text{expression} > \leftarrow (\{ < \text{term} > \}_0^\infty)$$

$$< \text{term} > \leftarrow (\text{quantity } \{ < \text{factor} > < \text{power} > \}_0^\infty)$$

$$< \text{factor} > \leftarrow \{ < \text{expression} > \}_0^\infty \text{ symbol}$$

$$< \text{power} > \leftarrow \text{quantity} \mid < \text{expression} > ,$$

where the left-hand side of each line represents the class of objects being defined; the angles < > enclose class names; braces { } indicate that the enclosed objects are to be included in the definition for each number of repetitions indicated by the numbers on the right brace; | separates alternative definitions; quantity is the class of floating point quantities other than zero; and symbol is the class of expression symbols.

In keeping with Bernstein's notation, double brackets will be used to denote real world expressions, and commas will be used between list elements.

Thus, the expression [[0]] may be represented as the list ( ) which is empty.

The expression [[x]] is a nonconstant expression consisting of one term with coefficient 1, a single factor 1HX, and the power 1. It is represented as the list ((1., 1HX, 1.)).

In general a Hollerith constant of from one to ten nonblank characters is used to represent each symbol in an expression. All function arguments are represented as lists. An expression is broken up into a sum of terms, where each term is broken up into a product of factors raised to powers. The trivial coefficient and power 1 are also introduced wherever necessary. Parentheses without a function name have been treated as the appearance of a special function name that has arbitrarily been translated into the expression symbol 3H1.*.

## B. Ordering and Simplification

In SYMBOLANG, identical terms differing only in their coefficients are combined, and a 0 coefficient causes a term to be dropped from an expression. Identical factors differing only in their powers are combined, and a 0 power causes a factor to be dropped. If the constant expressions [[0]] and [[1]] are raised to any power, they are unchanged.

SYMBOLANG imposes an ordering on all expressions and their components. All operations preserve this ordering, and some impose it. The ordering imposed is built up from orderings on numbers and expression symbols. Numbers are ordered arithmetically and expression symbols lexicographically using the following ordered alphabet:

```
0 1 2 3 4 5 6 7 8 9 A B C D E F G
H I J K L M N O P Q R S T U V W X
Y Z + - * / ( ) $ = ' . [ ] : ↑ < >
```

with blank coming before all other characters. Thus, 3H1.* is less than 1HA, which is less than 2HAA which is less than 1H$.

Factors within terms and terms within expressions are reordered to achieve expressions of the lowest possible ordering. Terms and factors are combined and dropped where possible as previously defined.

A hand created expression may be simplified by calling subroutine SMPL (one of the 72 SYMBOLANG subroutines).

SMPL (LIST) reorganizes the expression LIST into simplified form returning LIST as its value. The operation is timewise expensive and should be used sparingly.

## C. System Usage

The list of available space must be initialized. For this we can use:

```
DIMENSION WORK (5000)
CALL INITAS (WORK, 5000)
```

The call to INITAS also sets up a pointer and 100 public lists in blank COMMON. These are accessed by:

```
COMMON AVSL, X(100)
```

X(98), X(99), and X(100) are used extensively in SYMBOLANG and should not be altered by the user.

The system uses names of six characters beginning with LSQ, LTQ, XSQ, and XTQ so the user should not define routines or labeled common blocks with these characters.

In addition to the SYMBOLANG subroutines, the system requires many of the SLIP subroutines. For completeness, the author has made available the entire SLIP package for use with the system.

Following is a list of all the entry points in the SYMBOLANG-SLIP package.

| | | | | |
|---|---|---|---|---|
| ADD | ADNLWR | ADVDL | ADVDR | ADVLEL |
| ADVLER | ADVLL | ADVLNL | ADVLNR | ADVLR |
| ADVLWL | ADVLWR | ADVSEL | ADVSER | ADVSL |
| ADVSNL | ADVSNR | ADVSR | ADVSWL | ADVSWR |
| ALOAD | ALOG | ATAN | BOT | BREAK |
| CONT | COS | CP | CPYTRM | DELETE |
| DERROR | DLSRDR | DVSUM | DVTRM | EQUAL |
| EVALUE | EXP | GETCOE | HITENT | ICHR |
| ID | INHALT | INITAS | INITRD | INLIST |
| INLSTL | INLSTR | INSBST | INSUBT | INTARG |
| INTENT | INTGER | IRADLS | IRALST | IRARDR |
| ITSVAL | KOKE | LANORM | LCNTR | LDATVL |
| LDIF | LEXICO | LIST | LISTAV | LISTMT |
| LISTON | LNKL | LNKR | LOC | LOCARG |
| LOCT | LOFRDR | LOOKUP | LPNTR | LPURGE |
| LRDRCP | LRDROV | LSCMPR | LSPNTR | LSQADD |
| LSQAPL | LSQCDR | LSQCMP | LSQCNM | LSQCPY |
| LSQCXP | LSQDEF | LSQDES | LSQDSF | LSQERR |
| LSQGAR | LSQGNF | LSQGTM | LSQIDR | LSQINI |
| LSQLCO | LSQMEX | LSQMN1 | LSQMN2 | LSQMN3 |
| LSQMN4 | LSQMN5 | LSQMN7 | LSQMNL | LSQMTM |
| LSQNAM | LSQOU1 | LSQOU2 | LSQOU3 | LSQOU4 |
| LSQOU5 | LSQOU6 | LSQOU7 | LSQOU8 | LSQOUT |
| LSQPNT | LSQRAZ | LSQSBS | LSQTRC | LSQTYP |
| LSQUMN | LSQUN1 | LSQUN2 | LSQUN3 | LSQUN4 |
| LSQUN5 | LSQUN6 | LSQUN7 | LSQVAL | LSQVVL |
| LSSCPY | LSTEQL | LSTMRK | LSTPRO | LVLRV1 |
| LVLRVT | MADATR | MADLFT | MADNBT | MADNTP |
| MADOV | MADRGT | MAKEDL | MAN1 | MAN2 |
| MAN3 | MAN4 | MAN5 | MAN6 | MAN7 |

| | | | | |
|---|---|---|---|---|
| MANY | MONO | MRKLSS | MRKLST | MTDLST |
| MTLIST | NAMEDL | NAMTST | NEWBOT | NEWTOP |
| NEWVAL | NOATVL | NUCELL | NULSTL | NULSTR |
| NUMPY | NXTLFT | NXTRGT | PARMT | PARMT2 |
| POPBOT | POPMID | POPTOP | POWER | PRESRV |
| PRIPUT | PRLSTS | PUTLST | RCELL | RDLSTA |
| REED | REEL | RESTOR | RITEA | RITEF |
| RITEI | RITEIT | RITEO | SAME | SBST |
| SCHATL | SCHATR | SEQLL | SEQLR | SEQRDR |
| SEQSL | SEQSR | SETDIR | SETIND | SETRAY |
| SHIN | SIN | SMPL | SOLVE | SQIN |
| SQOUT | SREED | SRTRM | STRDIR | STRIND |
| SUB | SUBSBT | SUBST | SUBSTP | SUBT |
| SUMPY | TAN | TANH | TERM | TOP |
| TRCAL | TRUNC | TSTCON | VALARG | VISIT |
| ZERO | | | | |

The pattern of code in SYMBOLANG is usually to create operands, perform expression operations, and to erase unneeded operands.

SYMBOLANG relies heavily on calling functions as if they were subroutines, and using subroutines as if they were functions. In order to call a FORTRAN function on the BRLESC II computer, one must provide an extra argument for the value of the function. This argument does not need to be accessed by the user, just provided by him. For instance, the function XFUN(A, B) may be called as follows:

```
CALL XFUN (C, D, TEMP)
```

TEMP would contain the value of the function.

In using a subroutine as a function, the subroutine must have at least one argument. The value of the subroutine cannot, however, be assumed to be available for use.

Functions and subroutines are often deeply nested in SYMBOLANG.

## II. SYMBOLANG SUBROUTINES

### A. Basic Set

The subroutines of SYMBOLANG will be discussed alphabetically. This is arbitrary, but it overcomes artificial groupings according to function or complexity.

(1) FUNCTION ADD (LIST1, LIST2)

ADD is one of the older SYMBOLANG functions whose use should be avoided. It has a newer counterpart LSQADD which should be used instead.

ADD returns the result of adding expression LIST1 to expression LIST2. LIST1 is destroyed, and LIST2 is replaced by the sum.

(2) SUBROUTINE ALOAD (ARRAY, VAR1, VAR2, . . ., VARK), where K = 1,55.

ALOAD stores the machine address of the variables VARi into the array elements ARRAY(I). A call to ALOAD should be made to store addresses of arguments of any routine that uses INTARG, LOCARG, or VALARG. ALOAD has a variable length calling sequence. ALOAD returns the argument count in the event an argument is a floating point 999 (used where multiple entries are not provided). The argument count is common block MNYARG.

(3) BREAK (LIST1, SYMBOL, LIST2, LIST3)

BREAK is one of the older SYMBOLANG functions whose use should be avoided. It has a newer counterpart LSQTRC which should be used instead. BREAK adds to the expression LIST2 those terms of the expression LIST1 that contain the expression symbol SYMBOL and replaces LIST2 with this result. The remaining terms of LIST1 are added to the expression LIST3 (which is changed to this new value).

(4) CPYTRM (LIST1, LIST2)

CPYTRM is one of the older SYMBOLANG functions whose use should be avoided. CPYTRM removes the top element from list or expression LIST1 and places it on the bottom of list or expression LIST2. If LIST1 is empty, no change is made and CPYTRM returns 0; otherwise, CPYTRM returns 1. Correct ordering is not necessarily preserved, and changes to LIST1 may affect expressions of which it is a subexpression.

(5) DVSUM (LIST1, SYMBOL, LIST2)

DVSUM is one of the older SYMBOLANG functions whose use should be avoided. It adds to the expression LIST2 the result of dividing expression LIST1 by expression symbol SYMBOL. LIST2 is replaced with this result which is also returned as the value of the function.

(6) DVTRM (LIST1, SYMBOL)

DVTRM divides expression LIST1 by expression symbol SYMBOL.

(7) EVALUE (LIST1, SYMBOL 1, QUANT 1, . . ., SYMBOL 6, QUANT 6)

EVALUE is one of the older SYMBOLANG functions whose use should be avoided. A number of other routines are available for evaluation. EVALUE evaluates the expression LIST1 with the expression symbols SYMBOL i set to the floating point constants QUANTi. The evaluate returned as the value of the function also replaces the expression LIST1. EVALUE

has a fixed length calling sequence on BRLESC II. The floating point value of 999 should be used for SYMBOL i to signify that it and the arguments which follow it are not to be considered in the evaluations. Arguments following the 999 flag must be set to zero to insure proper calling sequence termination.

(8) GETCOE (SYMBOL, QUANT, LIST1, LIST2)

GETCOE is one of the older SYMBOLANG functions whose use should be avoided. It adds to the expression LIST2 the coefficient of the expression symbol SYMBOL raised to the floating point constant power QUANT in the expression LIST1, LIST2 is replaced with this expression which is also returned as the value of the function.

(9) HITENT (LIST1, SYMBOL, LIST2, LIST3)

HITENT also containing entry INTENT is one of the older SYMBOLANG functions whose use should be avoided. It returns the result of adding a level 1 substitution of the expression LIST1 for the expression symbol SYMBOL in the expression LIST2 to the expression LIST3. INTENT acts as does HITENT except that the substitution is performed on all levels. LSQSBS is a replacement for HITENT in the new code.

(10) ICHR (IALPHA)

ICHR returns a number for a letter of the alphabet or the digits 0-9. One is returned for A, 2 for B, . . ., 26 for Z, 27 for 0, . . ., and 36 for 9.

(11) INLIST (VAR, QUANT1, QUANT2)

INLIST returns a list representing the expression found on logical unit or list QUANT1. The variable or array element VAR will also contain the newly created expression. INLIST is a means of translating FORTRAN-like expressions into their internal representation on lists. Input is handled line by line with columns 7 through 72 scanned for an expression. If a $ is encountered, or the word NOMOR is encountered in columns 1-5, or if columns 1-5 are nonblank, the scan ends. Expressions are formed from symbols and numbers combined by the arithmetic operators +, -, *, /, and **, balanced parentheses, and commas.

On occasion, it is desirable to have the capability of inputting an equation. In this instance, if QUANT2 is set to 3HVAL, it is possible to input an equation, with the "=" being added to the operator list. When this option is desired, a double $ is needed to terminate the input expression. In all cases where an "=" appears and this option is not desired, QUANT2 should be set to the floating value of 999. In addition, the expressions before the last equal sign (in case there are more than 1)

are effectively comments. For instance, the input line of the form E = M*C**2 = 10$ is translated into the valid representation of [[10]]. Even though these expressions before the last equal sign are effectively comments, they must conform to the syntax for input expressions.

(12) INSBST (LIST1, SYMBOL, LIST2, LIST3)

INSBST is one of the older SYMBOLANG functions whose use should be avoided. It returns the result of adding an all-levels substitution of the expression LIST1 for the expression symbol SYMBOL in the expression LIST2 to the expression LIST3. LSQSBS is a better choice for this type of substitution.

(13) INSUBT (LIST1, SYMBOL, LIST2, LIST3, SYMBOL1, QUANT1, . . ., SYMBOL5, QUANT5)

INSUBT is a function of a fixed number of arguments. The argument list should be terminated by a floating point value of 999 for the first symbol SYMBOLi not used, followed by zeros for all remaining arguments. INSUBT returns the result of adding an all levels substitution of the expression LIST1 for the expression symbol SYMBOL in the expression LIST2 to the expression LIST3 and then truncating on the expression symbols SYMBOLi to the powers given by the floating-point quantities QUANTi. The expression LIST3 is replaced by the result. INSUBT is one of the older SYMBOLANG routines whose use should be avoided. LSQSBS and LSQTRC should be used to perform the operations described.

(14) INTARG (ARGS, INT)

INTARG returns the value of the INTth argument ARGS of the routine that called INTARG. A call of ALOAD must be made before INTARG may be used.

(15) KOKE (ALPHA)

KOKE stores ALPHA on an internal list and returns 0 if it has not encountered the last item to be stored and 1 if it has. KOKE is called by PUTLST an older SYMBOLANG function whose use should be avoided.

(16) LDIF (LIST, SYMBOL)

LDIF returns the derivative of expression LIST with respect to the expression symbol SYMBOL. Only explicit functional dependence is considered.

(17) LEXICO (LIST)

LEXICO is a lexicographical sort of the expression LIST.

(18) LISTON (VAR, QUANT1)

LISTON is a two argument call to INLIST for input on the standard input unit (at the BRL unit 5).

(19) LOCARG (ARGS, INT)

LOCARG returns the machine address of the INTth word of ARGS. A call of ALOAD must be made before LOCARG may be used.

(20) LOOKUP (ITYP, SYMBOL, LIST, INT)

LOOKUP defines the expression LIST to be the partial derivative of the function represented by the expression symbol SYMBOL, with respect to the argument INT, an integer quantity. ITYP is used when the function is determined by user definitions. If ITYP is not zero, a user-defined function is assumed.

(21) LSCMPR (LA, LB)

LSCMPR compares the terms LA and LB with their coefficients removed. Zero is returned if LA and LB are equal, -1 if LA is less than LB, and +1 if LA is greater than LB.

(22) LSQADD (LIST1, LIST2)

LSQADD returns the sum of expressions LIST1 and LIST2.

(23) LSQAPL (NAM, LIST, QUANT)

LSQAPL applies the function NAM to expression LIST and all sub-expressions of LIST if QUANT is any value except floating point 999.

(24) LSQCDR (LIST, SEQ)

LSQCDR substitutes a user-defined argument in expression LIST using sequence reader SEQ.

(25) LSQCMP (LA, LB, ITYPE)

LSQCMP is a lexicographical comparison of LA and LB. If ITYPE is 3HEXP then LA and LB are expressions, if ITYPE is 3HTER, they are terms, and if ITYPE is 3HFAC, they are factors. If ITYPE is anything else, LA and LB are terms to be compared as if their coefficients were 1.0. Note that the sequence readers are advanced if ITYPE is 3HFAC. Zero is returned if LA and LB are identical, -1 if LA is lexicographically less than LB, and +1 if LA is lexicographically greater than LB.

(26) LSQCNM (SYMBOL1, SYMBOL2)

LSQCNM returns 0 if the expression symbols SYMBOL 1 and SYMBOL 2 are identical, -1 if SYMBOL1 is lexicographically less than SYMBOL 2 and +1 if SYMBOL 1 is lexicographically greater than SYMBOL2.

(27) LSQCPY (SEQ)

LSQCPY returns a list which is a copy of the items to the right of the current setting of sequence reader SEQ.

(28) LSQCXP (LIST)

LSQCXP returns the list or expression LIST or a full copy of LIST via LSSCPY. The choice is determined by the setting of the variable LSQCSX in common block LSQCSX. If the variable is 0, the list itself is returned; otherwise, LSSCPY (LIST) is returned.

(29) LSQDEF (SYMBOL, LIST1, LIST2, NOCUR)

LSQDEF defines the symbol SYMBOL to be the expression LIST1 in which the expression symbols on the list LIST2 hold the places into which any arguments applied to SYMBOL are to be substituted. Both LIST1 and LIST2 are erased by the call. The user really uses LSQDEF to define his functions which may be evaluated later in his program.

When NOCUR is 5HNOCUR, the expression symbol SYMBOL is made to appear undefined. This permits SYMBOL to appear within LIST1 without causing infinite loops.

LSQDEF must be called with four arguments; however, the first argument which is an integer 999 effectively terminates the calling sequence. The result of the calling sequence in this instance should be padded out with zeros.

(30) LSQDES (LIST)

LSQDES erases the list or expression LIST. When a list is no longer needed, it should be destroyed to make cells available for other lists.

(31) LSQDSF (FUNCT, SYMBOL)

LSQDSF aids in defining and evaluating functions. The FORTRAN function FUNCT (defined in an external statement) is applied to the next expression, if any, reached by a right advance of LSQDMF(13). If this expression is constant, a constant is returned; otherwise, the expression symbol SYMBOL is used as a function name, and this expression found by the right advance of LSQDMF(13) as its argument.

(32) LSQERR (ARG)

LSQERR writes an error message and terminates the SYMBOLANG run. ARG is a Hollerith word giving the name of the routine in which the error occurred.

(33) LSQGAR (SYMBOL)

LSQGAR returns the next expression to be reached by a right advance of LSQDMF(13) if there is one. If not, an error is given with expression symbol SYMBOL as the error message. If SYMBOL = 0, a zero is returned if there is no next expression in the advance.

(34) LSQGNF (SEQ, NAME, NOARGS)

SEQ is a sequence reader assumed pointing to the first item of a factor within a term. SEQ is advanced until it points to a non sublist and the last value of the advancement flag is returned as the value of LSQGNF. NAME and NOARGS contain the name of the function in the factor and its number of arguments.

(35) LSQGTM (LA, VAR, POW1, POW2, LVAL)

See LSQTRC with a calling sequence of LA, VAR, POW1, POW2, LVAL +1, 1.

(36) LSQIDR

LSQIDR is a subroutine called by LDIF (on its first use) to define the derivatives of the expression symbols: 3H1.*, 3HSIN, 3HCOS, 3HTAN, 3HCOT, 3HSEC, 3HCSC, 6HARCSIN, 6 HARCCSC, 4HSINH, 4HCOSH, 4HTANH, 4HCOTH, 4HSECH, 4HCSCH, 7HARCSINH, 7HARCCOSH, 7HARCTANH, 7HARCCOTH, 7HARCSECH, 7HARCCSCH, 3HEXP, and 3HLOG. The defined derivatives are respectively: 1., COS(X), -SIN(X), SEC(X)**2, -CSC(X)**2, TAN(X)*SEC(X), -COT(X)*CSC(X) 1/(1 - X**2)**.5, -1/(1-X**2)**.5, 1/(1 + X**2), -1/(1 + X**2), X**-1*(X**2-1)**-.5, -X**-1*(X** 2-1) **-.5, COSH(X), SINH(X), SECH(X)**2, -CSCH(X)**2, -SECH(X)*TANH(X), -CSCH(X)*COTH(X), 1/(X**2 + 1)**.5, 1/(X**2-1)**.5, 1/(1-X**2), -1/(X**2-1), -X**-1*(1-X**2)**-.5, -X**-1*(X**2+1)**-.5, EXP(X), 1/X.

(37) LSQINI

LSQINI is a subroutine called by LSQVAL (on its first use) to define the expression symbols: 3HSIN, 3HCOS, 3HEXP, 3HLOG, 3HTAN, 6HARCTAN, 4HTANH, 3H1.*, 2HV., 2HQ. and 3HIF. The definitions represent respectively: SIN(X), COS(X), EXP(X), LOG(X), TAN(X), ARCTAN(X), TANH(X), the evaluates of the arguments enclosed in parentheses (3H1.*), the argument evaluated twice and returned as its value (2HV.), the unevaluated argument (2HQ.), and a conditional expression such that scanning from left to right, it returns the first even numbered argument following an odd numbered argument that evaluates to 0 (3HIF.). The value returned

is an unevaluated argument. If there is an odd number of arguments, the last argument is returned unevaluated. In all other cases, 0 is returned.

Note LSQINI does not define all of the functions whose derivatives are defined in LSQIDR.

(38) LSQLCO (LIST)

LSQLCO returns a lexicographically ordered copy of expression LIST.

(39) LSQMEX (LIST1, LIST2)

LSQMEX returns the product of expressions LIST1 and LIST2.

(40) LSQMNL (QUANT1, . . ., QUANT25)

LSQMNL returns a list on which its arguments QUANTi, i = 1, 25, appear. LSQMNL has entries LSQMN1, LSQMN2, LSQMN3, LSQMN4, LSQMN5, and LSQMN7 to account for calls with 1, 2, 3, 4, 5, and 7 arguments.

(41) LSQMTM (LIST1, LIST2)

LSQMTM returns the product of terms LIST1 and LIST2.

(42) LSQNAM (LNAM)

LSQNAM returns a unique name for each time it is called in its argument and for its value.

(43) LSQOUT (QUANT1, . . ., QUANT25)

LSQOUT outputs items for function LSQPNT. The argument list is indexed through and obeyed sequentially. LSQOUT has QUANTi, i = 1, 25 arguments. Entries LSQOU1, LSQOU2, LSQOU3, LSQOU4, LSQOU5, LSQOU6, LSQOU7, and LSQOU8, are provided to account for calls with 1, 2, 3, 4, 5, 6, 7, and 8 arguments respectively. The following argument blocks are defined:

| INITIAL ARGUMENT | NUMBER OF ARGUMENTS IN BLOCK |
|---|---|
| 5HFLUSH<br>Forces current line out. | 1 |
| 4HEDGE<br>Sets right margin | 2 |
| 6HMARGIN<br>Sets left margin to integer represented by second argument of the block. | 2 |

4HUNIT 2

Switches output to a new logical unit.

5HCFLAG 2 or 3

If second argument of the block is zero, no special continuation character is to be used (i.e., in initial state); otherwise, the second argument specifies a column to continue a continuation flag (usually 6) and the left most non-blank character of the third argument is used in that column for continuation images.

0 2

Outputs characters to a block or the end of a word.

Positive integer 2

Outputs the specified number of characters from the second argument which may spread over several machine words.

(44) LSQPNT (LIST, LTITLE, VALU)

LSQPNT is used to print expressions LIST and titles for expressions LTITLE. All other uses of LSQPNT are for internal use of SYMBOLANG. The third argument should be set to floating point 999.

(45) LSQRAZ (LIST1, LIST2)

LSQRAZ returns the result of raising the expression LIST1 to the expression LIST2 power.

(46) LSQSBS (LIST1, SYMBOL, LIST2, INT)

LSQSBS returns the result of substituting expression LIST1 for the expression symbol SYMBOL in the expression LIST2 through the level specified by the integer quantity INT. If INT is -1, a substitution is made on all levels (an infinite loop may occur if LIST1 contains SYMBOL).

(47) LSQTRC (LIST, SYMBOL, QUANT1, QUANT2, INT1, INT2)

LSQTRC returns an expression derived from expression LIST1 by retaining certain terms of the expression through level INT1, an integer quantity, and all terms on deeper levels. A term is retained if it contains the expression symbol SYMBOL to a nonconstant power or to a power between the floating point quantities QUANT1 and QUANT2. INT2 helps to determine how the truncation occurs. If INT2 is 1 or 2, one gets SYMBOL**EXPRESSION TRUNCATED. If INT2 is 2 or 4 one gets SYMBOL**P truncated, P between QUANT1 and QUANT2, and for INT of 1 or 3, one gets SYMBOL**P truncated with P not between QUANT1 and QUANT2.

(48) LSQTYP (QUANT, VAR)

LSQTYP returns -1 if the quantity QUANT is not a list and 0 or greater if it is a list or expression. The variable VAR is set to the quantity QUANT if QUANT is not a list. If QUANT is to be considered a list but not necessarily an expression, then VAR will not contain useful information.

(49) LSQUMN (LIST, QUANT1, . . ., QUANT25)

LSQUMN removes k (k = 1, 25) items from the bottom of list LIST, the bottom item going to QUANTi, then next to QUANTi-1, etc. The respective entries LSQUN1, LSQUN2, LSQUN3, LSQUN4, LSQUN5, LSQUN6, and LSQUN7 are provided to handle calls with 1, 2, 3, 4, 5, 6, or 7 arguments. It undoes the effect of MANY.

(50) LSQVAL (LIST, INT)

LSQVAL returns the result of evaluating expression LIST through level INT, an integer quantity. If INT is negative, no limit is in effect. If INT is 0, a copy of the expression LIST is returned. The one argument call (i.e., a call with INT = 999) is equivalent to a call with negative INT.

(51) LSQVVL (LIST)

LSQVVL returns a location to VISIT in order to evaluate the expression LIST. The scanning of arguments is made possible using LSQDMF(13) as a communication cell.

(52) MANY (LIST, QUANT1, . . ., QUANT25)

MANY places QUANTi, i = 1, 25, in turn to the bottom of LIST and returns LIST as its value. The entries MAN1, MAN2, MAN3, MAN4, MAN5, MAN6, and MAN7 are provided for call with 1, 2, 3, 4, 5, 6, and 7 arguments.

(53) MONO (QUANT)

MONO provides the quantity QUANT with an integer name.

(54) NUMPY (LIST, QUANT)

NUMPY is one of the older SYMBOLANG routines whose use should be avoided. It replaces the expression LIST with the result of multiplying this expression with the floating point constant QUANT and returns the product as its value.

(55) POPMID (VAR)

POPMID returns the list element on top of which the variable VAR is sitting, removes that element from the list, and advances VAR to

the left. VAR is treated as a sequence reader but should not be used if it was newly created.

(56) POWER (LIST, SYMBOL)

POWER returns the leftmost power to which the expression symbol SYMBOL is raised in the expression LIST, considering only the first level. Zero is returned if SYMBOL is not found.

(57) PRIPUT (HOL, LIST)

PRIPUT is one of the older SYMBOLANG routines whose use should be avoided. PRIPUT is equivalent to LSQPNT and is used for outputting.

(58) PUTLST (LIST, QUANT1, . . ., QUANT27)

PUTLST is one of the older SYMBOLANG routines whose use should be avoided. PUTLST returns the expression LIST onto which the terms QUANTi have been added. If PUTLST is called as above but with a leading 0 in the calling sequence, PUTLST will not reorder factors.

(59) SAME (QUANT)

SAME provides the quantity QUANT with a floating-point name.

(60) SBST (LIST1, SYMBOL, LIST2, LIST3)

SBST acts as does INSBST except that substitution is limited to one level.

(61) SETRAY (A, N, QUANT)

SETRAY stores the quantity QUANT into the array A whose dimension is N.

(62) SMPL (LIST)

SMPL reorganizes the expression LIST into simplified form, returning LIST as its value. The reorganization is an actual change and can properly alter an existing expression. The call is a costly one whose use should be made sparingly.

(63) SOLVE (LIST1, SYMBOL, LIST2)

SOLVE is one of the older SYMBOLANG routines whose use should be avoided. It assures that the expression LIST1 is linear in the expression symbol SYMBOL and solves the equation (LIST1 = 0) for SYMBOL. The solution is added to expression LIST 2 which is replaced by the sum. The sum is returned as the value of the function.

(64) SRTRM(L)

SRTRM replaces term L with term L sorted lexicographically.

(65) SUB(LIST1, LIST2)

SUB is one of the older SYMBOLANG routines whose use should be avoided. It returns the result of subtracting expression LIST1 from expression LIST2. LIST1 is destroyed and LIST2 is replaced with the difference.

(66) SUBT(LIST, SYMBOL, LIST2, LIST3, SYMBOL1, QUANT1, . . ., SYMBOL5, QUANT5)

SUBT acts as does INSUBT except that the substitution is limited to one level. If SYMBOLi is set to the floating point value of 999, it ends the effective calling sequence which should, however, be padded out with zeros.

(67) SUMPY (LIST1, LIST2, LIST3, SYMBOL1, QUANT1, . . ., SYMBOL5, QUANT5)

SUMPY is one of the older SYMBOLANG routines whose use should be avoided. It returns the result of adding the product of expressions LIST1 and LIST2 to expression LIST3 and truncating on the expression symbols SYMBOL i to the powers QUANTi. The first SYMBOLi set to floating point 999 terminates the effective calling sequence; however, the calling sequence should be padded out with zeros.

(68) TRCAL (LIST, SYMBOL1, SYMBOL2)

TRCAL is one of the older SYMBOLANG routines whose use should be avoided. It finds the lowest possible power to which the symbol SYMBOL2 is raised in those terms of the expression LIST that contain the expression symbol SYMBOL1. Then, the expression LIST is replaced by a truncated copy in which those terms that contain the expression symbol SYMBOL1 to a power greater than that minimal power of SYMBOL 2 are discarded.

(69) TRUNC (LIST, SYMBOL, QUANT)

TRUNC is one of the older SYMBOLANG routines whose use should be avoided. It replaces the expression LIST with the result of discarding all first level terms that contain the expression symbol SYMBOL to either a power greater than the floating point quantity QUANT or a non-constant power.

(70) TSTCON (LIST, VAR)

TSTCON returns 0 if the expression LIST is not a constant expression. If it is a constant, it returns 1 and the variable VAR is set to the equivalent FORTRAN constant value.

(71) VALARG (ARGS, INT)

VALARG is the same as INTARG except that it has a floating point name.

(72) ZERO (I, J)

ZERO zeros out J cells of array I.

B. Expanded Capability Set

In addition to the basic SYMBOLANG subroutines, several other routines are discussed by Dr. Bernstein. These routines are presented below.

(1) LSQCON (QUANT)

LSQCON returns an expression equivalent to the constant QUANT.

(2) LSQGCO (SYMBOL, QUANT, LIST)

LSQGCO returns the coefficient of the expression symbol SYMBOL raised to the power QUANT in the polynominal LIST.

(3) SUBROUTINE LSQIEQ

LSQIEQ causes 6HEQUAL to be defined as a means of creating definitions. 6HEQUAL arises with the three argument call of INLIST, where one wants to input an equation. LSQIEQ limits 6HEQUAL to two useful arguments.

(4) SUBROUTINE LSQIPN

LSQIPN defines the expression symbol 5HPRINT to print the value of its first argument with its second argument as a name.

(5) LSQNDR (LIST, SYMBOL, N)

LSQNDR returns the Nth derivative of expression LIST with respect to expression symbol SYMBOL.

(6) LSQNEB (LIST, N)

LSQNEB returns the Nth element from the bottom of list LIST.

(7) LSQNET (LIST, N)

LSQNET returns the Nth element from the top of list LIST.

(8) LSQXPN (LIST)

LSQXPN returns a version of the expression LIST in which parentheses have been largely removed. Parentheses remain only for expressions to positive or negative fractional powers and to the power -1 unless a non-constant power is encountered.

## III. PROGRAM EXAMPLES

### A. Input and Output

The following code may be used to input and output expressions:

```
      PROGRAM INPUT
      DIMENSION SP (500)
      CALL INITAS (SP, 500)
1     CONTINUE
      CALL INLIST (LA, 5HINPUT, 999, TEMP)
      CALL LSQPNT (LA, 2HLA, 999., TEMP)
      GO TO 1
      END
```

The following input cards run with the above program produce the output shown below.

Input

```
    A11$
    A12 + A11$

  NOMOR
```

Output

```
    A11$
    LA = A11
$
$ END OF EXPRESSION

    A12 + A11$
    LA = A11 + A12
$
$ END OF EXPRESSION

  NOMOR
```

```
    LA = 0
$
$ END OF EXPRESSION
```

An equation may be processed by changing the call to INLIST above as follows:

```
  CALL INLIST (LA, 5HINPUT, 3HVAL, TEMP)
```

The following input combined with this change produces the output below:

```
  E = M*C**2$$
  LA = EQUAL.  (E, C**2*M)
$
$ END OF EXPRESSION
```

Note that the $$ is needed to properly terminate a legitimate 3 argument call of INLIST. Note also the use of the cell TEMP with a call of a FORTRAN function.

## B. Arithmetic Calculation

The following program may be used to add and multiply expressions:

```
PROGRAM TEST
DIMENSION SP(500)
CALL INITAS (SP, 500)
CALL INLIST (LA, 5HINPUT, 999, TEMP)
CALL INLIST (LB, 5HINPUT, 999, TEMP)
CALL LSQPNT (LA, 2HLA, 999., TEMP)
CALL LSQPNT (LB, 2HLB, 999., TEMP)
LC = LSQADD (LA, LB)
LD = LSQMEX (LA, LB)
CALL LSQPNT (LC, 2HLC, 999., TEMP)
CALL LSQPNT (LD, 2HLD, 999., TEMP)
CALL EXIT
END
```

If combined with the input below, the ensuing output results:

```
INPUT
    A11$
    A12$
$ END INPUT
OUTPUT
    A11$
    A12$
```

```
      LA = A11

$
$ END OF EXPRESSION
      LB = A12
$
$ END OF EXPRESSION
      LC = A11 + A12
$
$ END OF EXPRESSION
      LD = A11 * A 12
$
$ END OF EXPRESSION
```

C. Differentiation

The following program allows one to differentiate expressions with respect to the expression symbol X.

```
      PROGRAM TEST
      DIMENSION SP (500)
      CALL INITAS (SP, 500)
1     CONTINUE
      CALL INLIST (LA, 5HINPUT, 999, TEMP)
      LB = LDIF(LA, 1HX)
      CALL LSQDES(LA, TEMP)
      CALL LSQPNT(LB, 5HDERIV, 999., TEMP)
      CALL LSQDES(LB, TEMP)
      GO TO 1
      END
```

Note the use of LSQDES to destroy unneeded expressions and thus reclaim storage.

The input below produces the output which follows.

```
INPUT
    1$
    X$
    X**2$
    X**N$
    1/X**N$
    SIN(A*X)/COS(A*X)$
    F(X,X,X)$
    LOG(N*X) - N*LOG(X)$

NOMOR

OUTPUT
```

```
      1$
      DERIV=0
    $
    $ END OF EXPRESSION
      X$
      DERIV=1
    $
    $ END OF EXPRESSION
      X**2$
      DERIV=2*X
    $
    $ END OF EXPRESSION
      X**N$
      DERIV=N*X**(-1+N)
    $
    $ END OF EXPRESSION
      1/X**N$
      DERIV=-N*X**(-1-N)
    $
    $ END OF EXPRESSION
      SIN(A*X)/COS(A*X)$
      DERIV=A + A*COS(A*X)**(-2)*SIN(A*X)**2
    $
    $ END OF EXPRESSION
      F(X,X,X)$
      DERIV=PARTIAL(N.0,1) + PARTIAL(N.0,2) + PARTIAL(N.0,3)
    $
      N.0 = F(X,X,X)
    $
    $ END OF EXPRESSION
      LOG(N*X) - N LOG(X)$
      DERIV=-N*X**(-1) + X**(-1)
    $
    $ END OF EXPRESSION

NOMOR
      DERIV=0
    $
    $ END OF EXPRESSION
```

D. Truncation

The following program may be used to truncate a programmed expression:

```
PROGRAM TEST
DIMENSION SP(500)
CALL INITAS(SP, 500)
LA = LSQMN3(LSQMN1(3.), LSQMN3(5., 1HX, 11.),
```

```
     1    LSQMN5(-4., 1HX, 22., 1HY, 1.))
          LA1 = LSQTRC(LA, 1HX, 0., 999., 0,0)
          LA2 = LSQTRC(LA, 1HY, 0., 999., 0,0)
          LA3 = LSQTRC(LA, 1HZ, 0., 999., 0,0)
          LA4 = LSQTRC(LA, 1HX, 11., 999., 0,0)
          CALL LSQPNT(LA, 2HLA, 999., TEMP)
          CALL LSQPNT(LA1, 3HLA1, 999., TEMP)
          CALL LSQPNT(LA2, 3HLA2, 999., TEMP)
          CALL LSQPNT(LA3, 3HLA3, 999., TEMP)
          CALL LSQPNT(LA4, 3HLA4, 999., TEMP)
          CALL EXIT
          END
```

The program above produces the following output:

```
          LA = 3 + 5*X**11 - 4*X**22*Y
          $
          $ END OF EXPRESSION
            LA1 = 3
          $
          $ END OF EXPRESSION
            LA2 = 3 + 5*X**11
          $
          $ END OF EXPRESSION
            LA3 = 3 + 5*X**11 - 4*X**22*Y
          $
          $ END OF EXPRESSION
            LA4 = 3 + 5*X**11
          $
          $ END OF EXPRESSION
```

E. Evaluation

The following program redefines a definition and simplifies an expression:

```
          PROGRAM TEST
          DIMENSION SP(1000)
          CALL INITAS (SP, 1000)
          LA = LIST(9)
          CALL LSQDES (LSQVAL(LA, 999), TEMP)
          CALL LSQDES (LA, 999)
          CALL LSQDEF (3H1.*, LSQMN1 (LSQMN3
        1 (1., 10H$.1.*$.$.$, 1.)), LSQMN1 (10H$.1. *$.$.$), 999, TEMP)
          CALL LSQDEF (3HCOS, LSQMN1(LSQMN4(1., LSQMN2(LSQMN1(1.),
        1 LSQMN3(-1., 3HSIN, 2.)), 3H.1*,.5)), 0, 999 TEMP)
       10 CALL INLIST (LA, 5HINPUT, 999, TEMP)
          IF (LISTMT(LA) .EQ.0) CALL EXIT
          CALL LSQPNT (LA, 5HINPUT, 999, TEMP)
          LB = LSQVAL (LA, 999)
```

```
      CALL LSQDES (LA, TEMP)
      CALL LSQPNT (LB, 5HVALUE 1, 999., TEMP)
      LC =  LSQVAL (LB, 999)
      CALL LSQDES (LB, TEMP)
      CALL LSQPNT (LC, 6HVALUE2, 999., TEMP)
      CALL LSQDES (LC, TEMP)
      GO TO 10
      END
```

The following inputs for the above program produce the output shown below.

```
INPUT
    COS**2 + SIN**2$
    COS(X)**2 + SIN(X)**2$
    COS(X)*SIN(X)$
OUTPUT
    COS**2 + SIN**2$
    INPUT = COS**2 + SIN**2
  $
  $ END OF EXPRESSION
    VALUE1 = SIN**2 + (1 - SIN**2)
  $
  $ END OF EXPRESSION
    VALUE2 = 1
  $
  $ END OF EXPRESSION
      COS(X)**2 + SIN(X)**2$
    INPUT = COS(X)**2 + SIN(X)**2
  $
  $ END OF EXPRESSION
    VALUE1 = SIN(X)**2 + (1 - SIN**2, X)
  $
  $ END OF EXPRESSION
    VALUE2 = 1
  $
  $ END OF EXPRESSION
      COS(X)*SIN(X)$
    INPUT = COS(X)*SIN(X)
  $
  $ END OF EXPRESSION
    VALUE1 = SIN(X)*(1 - SIN**2, X)**5E-1
  $
  $ END OF EXPRESSION
    VALUE2  = (1 - SIN(X)**2)**5E-1*SIN(X)
  $
  $ END OF EXPRESSION
```

## IV. CONCLUDING REMARKS

The full power of SYMBOLANG becomes apparent after one begins to use it. The subprograms of SYMBOLANG combined with those of SLIP give the user a wide variety of options for solving a large class of problems. In addition, one has at his disposal all of the FORTRAN language.

SYMBOLANG may be expanded by the user to include functions and subroutines of a general or specific nature. One such set of possible expansions is an integration package (see e.g. Slagle, J. R., A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus, Journal of the Association for Computing Machinery, 1963, 10, 507-520). The user also has at his disposal through SYMBOLANG the capability of defining his own functions and incorporating them into the system.

DISTRIBUTION LIST

| No. of Copies | Organization |
|---|---|
| 12 | Commander<br>Defense Documentation Center<br>ATTN: DDC-TCA<br>Cameron Station<br>Alexandria, Virginia 22314 |
| 1 | Commander<br>U.S. Army Materiel Command<br>ATTN: AMCDL<br>5001 Eisenhower Avenue<br>Alexandria, Virginia 22333 |
| 1 | Commander<br>U.S. Army Materiel Command<br>ATTN: AMCRD, BG H. A. Griffith<br>5001 Eisenhower Avenue<br>Alexandria, Virginia 22333 |
| 1 | Commander<br>U.S. Army Materiel Command<br>ATTN: AMCRD-T<br>5001 Eisenhower Avenue<br>Alexandria, Virginia 22333 |
| 1 | Commander<br>U.S. Army Aviation Systems Command<br>ATTN: AMSAV-E<br>12th & Spruce Streets<br>St. Louis, Missouri 63166 |
| 1 | Director<br>U.S. Army Air Mobility Research and Development Laboratory<br>Ames Research Center<br>Moffett Field, California 94035 |
| 1 | Commander<br>U.S. Army Electronics Command<br>ATTN: AMSEL-RD<br>Fort Monmouth, New Jersey 07703 |
| 1 | Commander<br>U.S. Army Missile Command<br>ATTN: AMSMI-R<br>Redstone Arsenal, Alabama 35809 |
| 1 | Commander<br>U.S. Army Tank Automotive Command<br>ATTN: AMSTA-RHFL<br>Warren, Michigan 48090 |
| 2 | Commander<br>U.S. Army Mobility Equipment Research & Development Center<br>ATTN: Tech Docu Cen, Bldg. 315<br>AMSME-RZT<br>Fort Belvoir, Virginia 22060 |
| 1 | Commander<br>U.S. Army Armament Command<br>Rock Island, Illinois 61202 |
| 1 | Commander<br>U.S. Army Harry Diamond Laboratories<br>ATTN: AMXDO-TI<br>Washington, DC 20438 |
| 1 | Director<br>National Bureau of Standards<br>Department of Commerce<br>Washington, DC 20234 |
| | Aberdeen Proving Ground |
| | Marine Corps Ln Ofc<br>Dir, USAMSAA<br>ATTN: J. Sperrazza<br>L. Bain<br>E. Belbot<br>W. Wenger |