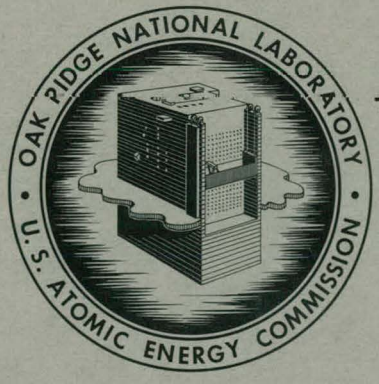
ORNL-3399
UC-32 – Mathematics and Computers
TID-4500 (18th ed., Rev.)

AN ALGOL 60 SYNTAX CHECKER FOR THE IBM 7090 COMPUTER

M. P. Lietzke



OAK RIDGE NATIONAL LABORATORY
operated by
UNION CARBIDE CORPORATION
for the
U.S. ATOMIC ENERGY COMMISSION

DISCLAIMER

Printed in USA. Price: $0.75 Available from the
Office of Technical Services
U. S. Department of Commerce
Washington 25, D. C.

ORNL-3399

Contract No. W-7405-eng-26

Mathematics Division

AN ALGOL 60 SYNTAX CHECKER FOR THE IBM 7090 COMPUTER

M. P. Lietzke

DATE ISSUED

MAR 21 1963

OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee
operated by
UNION CARBIDE CORPORATION
for the
U.S. ATOMIC ENERGY COMMISSION

THIS PAGE

WAS INTENTIONALLY

LEFT BLANK

CONTENTS

AN ALGOL 60 SYNTAX CHECKER FOR THE IBM 7090 COMPUTER

Marjorie P. Lietzke

ABSTRACT

A syntax checker was designed based on the syntax of Algol as described in the Algol 60 Report (ACM Communications, May, 1960). Since the definition of the elements of the language is recursive it was found most desirable to design the syntax checker as a set of mutually recursive processors tied together by building blocks which perform certain bookkeeping functions. Because of the recursive nature of the language and of the syntax checker the problem of recovery after an error required much attention. A method was devised which permits most programs to be checked completely despite errors. The syntax checker has been implemented for the IBM 7090 as a part of the SHARE ALGOL processor, and has operated very satisfactorily.

INTRODUCTION

The syntax checker for the SHARE Algol 60 Processor is based on the syntax of Algol 60 as described in the official report.(1) Only those restrictions which were necessary to make the syntax checker compatible with the internal translator written at IBM were included. As the restrictions are removed from the translator, they will also be removed from the syntax checker.

The elements of the Algol language are recursive in nature, i.e., an expression of any type may contain within itself expressions of the

---

[1] Report on the Algorithmic Language ALGØL 60, Peter Naur, et al, ACM Communications (May 1960).

same type or of other types. An arithmetic expression, for example, may contain a conditional expression, which in turn contains a Boolean expression. The Boolean expression may consist of two or more arithmetic expressions connected by relational operators. This recursive process may continue to any level, with resulting expressions entirely too complex to consider as a whole.

The only feasible method of syntax checking something this complex seems to be by breaking the large complex expressions down into the basic syntactic units and examining each unit as it is encountered. This, of course, immediately raises the problem of checking an expression within another expression of the same type. The direct result of this is a set of mutually resursive processing subroutines, one for each syntactic unit of the Algol language. Each processor is capable of calling itself or any other processor as necessary. In addition to the processing subroutines, it was necessary to design a group of subroutines which perform the operation required by the processors and take care of necessary bookkeeping. To distinguish between these two different types of subroutines they are referred to as processors and building blocks, respectively, in the balance of this report.

DESCRIPTION OF PROCESSORS

The processors are sets of tables which list the possible or expected symbols for each state in a syntactic unit. Each symbol has associated with it an "action" and a "next state". The "action" depends on the symbol and the state of the syntactic unit being checked and may range from no operation to a fairly complex sequence of operations,

including the call of other processors. The "next state" also depends on the current symbol and state and may depend on whether a processor invoked in the "action" part has given a normal return or an error return. The "next state" is always another state in the same processor, a return, or an error return.

The arithmetic expression processor shown in Table 1 is an example of a typical processor. When this processor is called, the assumption is that the first symbol of an arithmetic expression is under scan and checking begins at state AREX1. The symbol under scan is compared with each symbol in the table until a match is found or until the entry "other", meaning any other symbol, is encountered. At this point, the indicated action is taken and control passes to the indicated next state where comparison is resumed.

The processing of a syntactically correct arithmetic expression is shown in Figure 1. This example is not a complete statement, but an expression which could occur as part of a complete statement. It in turn contains expressions thus serving to illustrate the need for and use of recursive processors.

A complete list of the syntactic units for which processors were designed together with the code name of the processor is given in Table 2. The complete set of tables for the processors has not been included in this report since the program listing is completely annotated. The machine code for the processor given in Table 1 is shown in Figure 2.

In general, each of these processors can call directly only the processors for syntactic units which may properly be used within the unit

being checked. The block processor, for example, may call the general declaration processor or the general statement processor. It may not call directly an expression processor or a processor for a particular type of statement or declaration. The only exceptions to this rule are the resume and resume 2 processors. These processors were designed to continue the syntax check after an error has occurred. They operate together or separately to recognize and check any syntactic unit. After each complete unit has been checked by the resume or resume 2 processor control is returned to the processor in which the error was detected. If possible, normal scanning is continued. If the symbol under scan is still incorrect in context, the resume processors are re-entered and the next unit is scanned. This process is continued until the end of the program has been reached.

In actual testing, this method of continuing the checking has worked very well. Several redundant error messages may be produced, but it is usually obvious that they all resulted from the same error. In most cases, the entire program is scanned and all errors detected; only in programs which contain an extremely high percentage of errors does this method sometimes overlook some errors. Error messages given by the syntax checker are listed as comments on tape which is to be input to FAP, together with the ten source program symbols which precede the point at which the error was detected.

DESCRIPTION OF BUILDING BLOCKS

The building blocks for the syntax checker are essentially bookkeeping routines. In using the processors recursively, for example,

it is necessary to keep track of each call of a processor in order to make a proper return when processing has been completed. It is also nccessary to save the current values of certain quantities on entering a processor and to retrieve these values on leaving the processor. These functions are performed by five building blocks; recursive entry, recursive return, recursive error return, push up, and pull down, which operate on two push down lists. The state pushdown list contains the ordered list of the location of recursive entries to processors and the auxiliary pushdown contains the saved values of quantities used by the processors.

A complete list of the building blocks together with a brief statement of the function of each appears in Table 3. Detailed algorithms for each of the building blocks are given in Appendix A.

Table 1

Arithmetic Expression Processor

State	Symbol	Action	Next State: No Error	Next State: Error
AREX 1	+	skip (pass on this symbol, get next symbol)	AREX 2	
	-	skip	AREX 2	
	if	skip, Boolean Expression Processor	AREX 5	AREX 9
AREX 2	ID	get type of identifier	AREX 8	
	(	skip, Arithmetic Expression Processor	AREX 3	AREX 10
	other	———	Error Return	
AREX 3	)	skip	AREX 4	
	other	Error message	AREX 10	
AREX 4	+	skip	AREX 2	
	-	skip	AREX 2	
	×	skip	AREX 2	
	/	Real Exp: = true, skip	AREX 2	
	÷	skip	AREX 2	
	↑	skip	AREX 2	
	other	———	Return	
AREX 5	then	skip	AREX 6	
	other	———	AREX 9	
AREX 6	if	Error message, Insert (, Arithmetic Expression Processor, insert)	AREX 7	AREX 9
	other	Arithmetic Expression Processor	AREX 7	AREX 9
AREX 7	else	skip, Arithmetic Expression Processor	return	error return
	other	Error message	AREX 9	
AREX 8	real variable	real exp: = true, skip	AREX 4	
	integer variable	skip	AREX 4	

Table 1 (continued)

Arithmetic Expression Processor

State	Symbol	Action	Next State: No Error	Next State: Error
	array	Real exp: = true, skip, subscript processor	AREX 4	
	real array	Real exp: = true, skip, subscript processor	AREX 4	error return
	integer array	skip, subscript processor	AREX 4	error return
	Boolean variable	Error message, skip	AREX 4	
	Boolean array	Error message, skip, subscript processor	AREX 4	error return
	Boolean procedure	Error message, skip, function statement processor	AREX 4	error return
	other	Error message	error return	
AREX 9	then		AREX 5	
	else		AREX 7	
	other	Resume processor	AREX 9	error return
AREX 10	)		AREX 3	
	other	Resume processor	AREX 10	error return

Table 2

Processors for Syntactic Units of Algol

Syntactic Unit	Name of Processor
arithmetic variable	ARVA
truth value	TRUVA
subscripted variable	SUBV1
arithmetic expression	AREX1
general expression	GEX 0
Boolean expression	BOLEX
function	FUNC1
standard function	BIFTBL
arithemtic comparison	COMP1
label processor	LABEL
simple designational expression	SIDEX
designational expression	DESEX
statement	STAT0
assignment statement	AST1
go to statement	GTST
procedure statement	PROCD
for statement	FSTT
conditional statement	CNST
unconditional statement	UNCS
compound statement	CPST
block	BLOCK
comment	CMNT1
parameter list	PARLST

Table 2 (continued)

Processors for Syntactic Units of Algol

Syntactic Unit	Name of Processor
declaration	DECLO
type declaration	TYDL
array declaration	ARDL
switch declaration	SWIDL
procedure declaration	PRODL
function declaration	FUDL
specifier	SPCFR
parameter delimiter	PADEL
check parameter type	PARTYP
resume	RESUM
resume 2	RSMII
statement end	STEND
type	TYPE
switch variable declaration	SWVD
name list declaration	NAMD
iteration list processor	ITLST
library procedure declaration	LBPR

Table 3

Building Blocks for Algol Syntax Checker

Name	Purpose
REC	Calls any processor recursively, makes entry in State Push Down list.
INTERP	Transfers control on same recursion level within processors.
RETURN	Returns control recursively to calling processor when syntactic unit is correct.
ERRRT	Returns control when syntactic unit is incorrect.
SKIP	Gets next symbol to check, puts previous symbol away for translator.
PUT1	Puts away symbols for translation, gets no new symbols.
QULFY	Determines type of identifier from declaratory information previously processed.
GET1	Gets N^{th} symbol beyond one now under scan without disturbing symbols between.
DECLAR	Puts declaratory information in tables for future reference.
FOXL	Check labels declared in FOR statement and take care of necessary bookkeeping on leaving for loop.
ADVNCE	Gets next symbol for checking without transmitting previous symbol.
UP	Puts variables in auxiliary pushdown and increases counters.
DOWN	Retrieves variables from auxiliary pushdown as necessary and decreases counters.
QULFY1	Determines whether identifier whose type has been found by QULFY is a constant or a variable.
DARRAY	Puts declaratory information about arrays in tables, including subscript count.
DPARAM	Puts specifier information about parameters in tables for future use.

Table 3 (continued)

Building Blocks for Algol Syntax Checker

Name	Purpose
CMSPEC	Compress and store specifier and value information about
PARPOS	Find position (1st, 2nd, etc.) of parameter for value and specifier entries.
PARCNT	Check number of parameters used in function or procedure call.
FILA	Find or generate entry in label table for label under scan.
DELAB	Make entry in label table when a labelled statement is encountered.
LABUS	Make entry in label table when label is found in go to or in switch declaration.
FORLA	Take care of necessary bookkeeping on entering for loop.
BENGEL	Take care of necessary bookkeeping on entering Block.
BEXGEL	Take care of necessary bookkeeping on leaving Block.

Table 4

Error Messages from Syntax Checker

Error	Message	Action
ERR 1	state push-down exceeded. No compilation.	stop
2	identifier not declared. Treated as real.	
3	last identifier wrong type.	-
4	an error has occurred which makes last symbol illegal.	-
5	number of subscripts used different from No. declared.	-
6	auxiliary push-down list exceeded. No compilation.	stop
7	conditional expression after then. Begin, end inserted.	correct
8	parenthesis and bracket interchanged. Corrected	correct
09	array used as a simple variable	-
10	wrong type of expression?	-
11	statement incorrectly written	-
12	conditional statement not allowed. Corrected.	correct
13	expression wrong type.	-
14	declarations out of order. Compilation will continue.	-
15	semicolon followed by else. Corrected.	correct
16	ARRAY pushdown exceeded. No compilation.	
17	type own instead of own type. Corrected.	correct
18	declaration own x, y, ... Changed to own real x, y, ...	correct
19	possible machine error or error in translator	stop
20	label declared twice in same block	-
21	go to L occurred outside for statement containing L	-
22	label table pushdown exceeded. No compilation.	stop
23	processor expects variable name	-

Table 4 (continued)

Error Messages from Syntax Checker

Error	Message	Action
24	the following switch labels occur inside for loops	
25	L1 L2 LN	
26	may this lead to an illegal transfer.	
27	local variable used in an array declaration.	-
28	the following quantity was declared twice in 1 block	-
29	parameter table exceeded. No compilation.	-
30	value parameter not specified	-
31	value or specified parameter not in parameter list	-
32	parameter wrong type	-
33	incorrect number of parameters in procedure call.	-
37	identifier missing	
42	begins and ends do not match	
43	this use of array correct only as parameter	
ERR44	parameter specified twice	
ERR45	this parameter will not be dynamic	

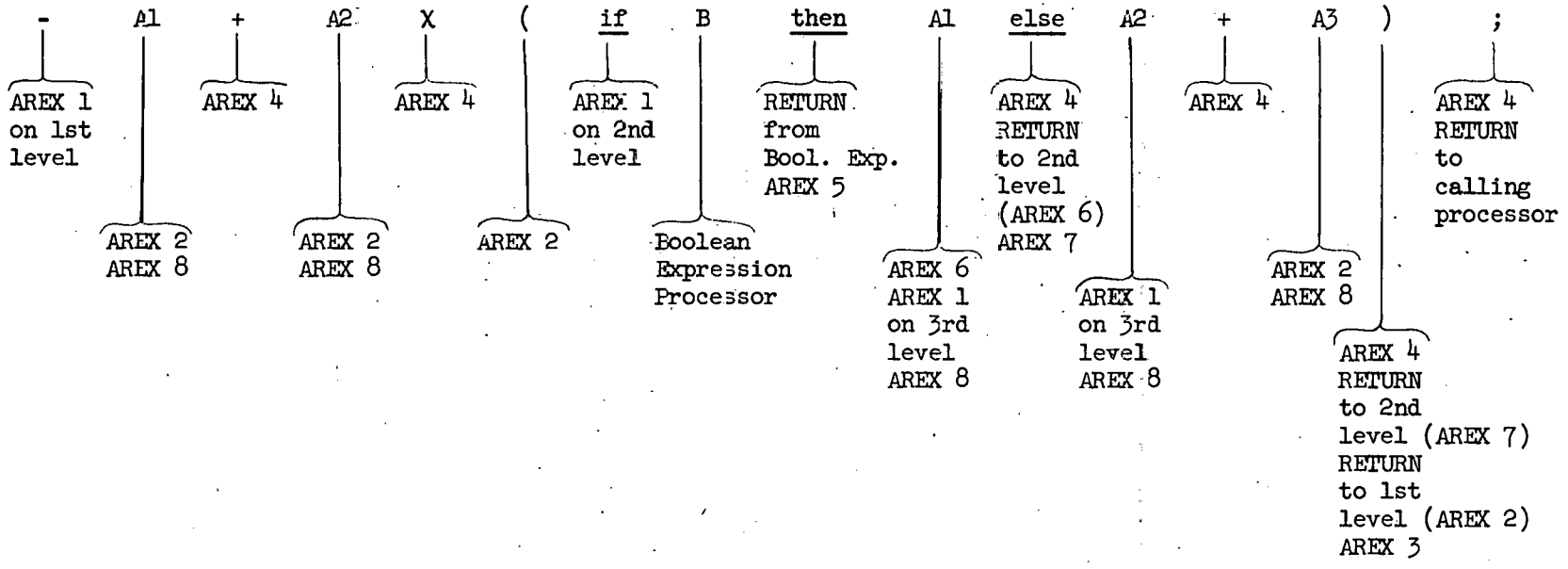


Figure 1. Processing of An Arithmetic Expression

FIGURE 2. ARITHMETIC EXPRESSION PROCESSOR

```
AREX1   PZE     )PLUS,,A1           SKIP..AREX2
        PZE     )UNARY,,A1          SKIP..AREX2
        PZE     )IF,,A2             SKIP,BOLEX..AREX5..AREX9
AREX2   PZE     )ID,,A4             QUALIFY..AREX8
        PZE     )LPAR,,A5           SKIP,AREX..AREX3..AREX10
        MZE     0,,A6               ..ERRRT
AREX3   PZE     )RPAR,,A7           SKIP..AREX4
        MZE     0,,A8               ERR4..AREX10
AREX4   PZE     )PLUS,,A9           SKIP..AREX2
        PZE     )MINUS,,A9          SKIP..AREX2
        PZE     )STAR,A9            SKIP..AREX2
        PZE     )SLASH,,A9A         REALEX=1, SKIP..AREX2
        PZE     )DIV,,A9            SKIP..AREX2
        PZE     )ARROW,,A9          SKIP..AREX2
        MZE     0,,A10              ..RETURN
AREX5   PZE     )THEN,,A11          SKIP..AREX6
        MZE     0,,A12              ..AREX9
AREX6   PZE     )IF,,A13            ERR7,AREX1..AREX7..AREX9
        MZE     0,,A14              AREX..AREX7
AREX7   PZE     )ELSE,,A15          SKIP,AREX..RETURN..ERRRT
        MZE     0,,A16              ERR4..AREX9
AREX8   PZE     )REALV,,A17A        REALEX=1,SKIP..AREX4..ERRRT
        PZE     )INTEG,,A17         SKIP..AREX4
        PZE     )ARRA,A18A          REALEX=1,SKIP,SUBV1..AREX4..ERRRT
        PZE     )RARRA,,A18A        REALEX=1,SKIP,SUBV1..AREX4..ERRRT
        PZE     )IARRA,,A18         SKIP,SUBV..AREX4..ERRRT
        PZE     )RPROC,,A19A        REALEX=1,SKIP,FUNC1..AREX4..ERRRT
        PZE     )IPROC,,A19         FUNCTION..AREX4..ERRRT
        PZE     )BOOLV,,A26         ERR3,SKIP..AREX4
        PZE     )BARRA,,A27         ERR3,SKIP,SUBV..AREX4..ERRRT
        PZE     )BPROC,,A28         ERR3,FUNCTION..AREX4..ERRRT
        MZE     0,,A20              ERR3..ERRRT
AREX9   PZE     )THEN,,A21          ..AREX5
        PZE     )ELSE,,A22          ..AREX7
        MZE     0,,A23              RESUME..AREX9..ERRRT
AREX10  PZE     )RPAR,,A24          ..AREX3
        MZE     0,,A25              RESUME..AREX10..ERRRT
```

APPENDIX A

ALGORITHMS FOR BUILDING BLOCKS

```
Comment the quantities declared below
enter all building blocks as globals;

Integer SPDPT, CLASS, SYMBOL, Bupoi, Loceop, LOCBOP, ID, NSl, BL, LATAB,
LATAE, TYPLOC, TYP, DECPTB, NS, KKK;

Boolean First Entry, NMSOT, ARD, SWDL;

Array State Push Down [1 : 50]
      Primary Table [1024 : 4096],
            Pritab type bits, Pritab Pointer [1024 : 4096],
      Declaration Table [1 : 1000],
            DEPTH1, DEPTH2, ISYMB, Error Count,
            ARG [1 : 1000],
      Buffer [1 : 10],
      Symbol Stack [1 : 100],
      TTT [1 : 400],
      Label Table [1 : 400],
          NAME [1 : 400],
          STAT [1 : 400],
          SWBT [1 : 400],
          FLT [1 : 400];
```

```
Procedure INTERP (Symbol list, Action); Array Symbol list;
Switch Action;
    Begin Integer I;
    For I := 0, 1+1 do
        begin
            if Symbol list [I] = ØTHER then
                go to Action [I] else
                if Symbol list [I] = CLASS then
                go to Action [I];
        end
end Interp;

Procedure REC (location, Symbol list, Action); Value location;
   label location; array Symbol list; switch Action;
      Begin State Push Down [SPDPT] := location;
            SPDPT := SPDPT + 1;
            if SPDPT > 50 then begin
               ERMESS (ERR1);
               go to Badend end State Push Down Exceeded
              else INTERP (Symbol list, Action)
      end REC;

Procedure  RETURN (location); label location;
     Begin SPDPT := SPDPT - 1;
           location := State Push Down [SPDPT] + 4;
           go to location
     end RETURN;
```

```
Procedure ERRRT (location); label location;
     Begin SPDPT := SPDPT - 1;
           location := State Push Down [SPDPT] + 2;
           go to location
     end ERRRT

Procedure SKIP; Comment GETIAL and PUTIAL
     are procedures to get a symbol from tape or put it on tape;
     Begin if First Entry go to SKIP1 else
  begin Buffer [Bupoi] := SYMBOL
        Bupoi := Bupoi + 1;
        if Bupoi > 10 then Bupoi := 1;
        PUTIAL (SYMBOL);
  end;
SKIP1: if NMSOT then go to SKIP3 else
  begin GETIAL (SYMBOL, SKIP2);
        Loceop := Loceop + 1;
        Symbol Stack [Loceop] := SYMBOL;
        if Loceop > 99 then Loceop := 1;
        go to SKIP3 P6
  end;
SKIP3:    if Locbop - Loceop = 1 then go to EOFP;
SKIP3P6:  SYMBOL := CLASS := Symbol Stack [Locbop]
          Locbop := Locbop + 1;
          if Locbop > 100 then Locbop = 1;
          if Symbol ≥ 1024 then go to SKIP4;
```

```
SKIPRT:   go to if First Entry then SKIP5 else SKIPEND;
SKIP4:    CLASS :=ID;
          go to SKIPRT;
SKIP2:    NMSOT := true;
          go to SKIP3;
SKIP3:    FIRST ENTRY :=false;
SKIPEND:  end SKIP;

Procedure PUT1 (ARG); Array ARG;
     begin Integer I;
         for I := 1, I+1 do
        begin PUTIAL (ARG [I])
            if ARG [I] < 0 then go to PUTRT
        end;
     PUTRT: end PUT1;

Procedure QULFY;
     Begin
        Typ = Pritab type bits [SYMBOL];
        if typ = 0 then go to QULF2;
        if typ < 8 then begin class := class + Typ;
                              go to QULF3
                        end;
          class := class + Type;
          NS1 := Argument [Pritab Pointer [SYMBOL]];
```

```
QULF3:      if ARD then
    begin if DEPTH1 [Pritab Pointer [SYMBOL]] = BL then
            begin ERMESS (ERR 27);
                  go to QULFYRT
            end Local variable in array declaration
            else DEPTH2 [Pritab pointer [SYMBOL]] = BL
    end
            else go to QULFRT;
QULF2:      ERMESS (ERR2);
            Comment Undeclared variable, treated as real;
            Pritab type bits [SYMBOL] := 3;
            DEPTH1 [Pritab Pointer [SYMBOL]] := BL;
            DEPTH2 [Pritab Pointer [SYMBOL]] :=
            PSYMB [Pritab Pointer [SYMBOL]] :=
            Error Count [Pritab Pointer [SYMBOL]] :-
            ARC [Pritab Pointer [SYMBOL]] :- 0;
            go to QULF3;
QULFRT:        end QULFY;

Procedure  GET1 (N, L); Value N; Integer N;
            label L;
Begin if N < 99 then go to L;
      N := N + Locbop - 1;
      if N = 0 then N := 100;
      if N > 100 then N:= N - 100;
      if NMSOT then go to GETIA
```

```
GET1B:   CLASS := SYMBOL := Symbol Stack[N];
         if SYMBOL < 1024 then go to GETRT;
         CLASS := ID; go to GETRT;
GET1A: if Locbop - Loceop < 0 then begin
       if Locbop - Loceop + 99 > N then
       go to GET1B else go to L end
     else if Locbop - Loceop > N then
          go to GET1B else go to L;
GETRT:   end GET1;

Procedure Declar;
Begin if Pritab type bits [SYMBOL] = 0 then go to NYD;
      if BL = DEPTH1 [Pritab Pointer [SYMBOL]] then go to
         CLAR28;
      if BL = DEPTH2 [Pritab Pointer [SYMBOL]] then go to
         CLAR27;
NYD:  DECPTB := DECPTB + 1;
      Pritab Pointer [SYMBOL] := DECPTB;
      DEPTH1 [Pritab Pointer [SYMBOL]] := BL;
      DEPTH2 [Pritab Pointer [SYMBOL]] :=
      PSYMB [Pritab Pointer [SYMBOL]] :=
      ARG [Pritab Pointer [SYMBOL]] := 0;
      if   ERROR then
      Error Count [Pritab Pointer [SYMBOL]] := 0;
      Pritab type bits [SYMBOL] := TYPLOC;
```

```
        If typloc < 8 V typloc > 11 then go to DRET
        else ARG [Pritab Pointer [symbol]] := NS;
        go to DRET;
CLAR 27:  ERMESS (ERR27);
CLAR27P2 Error Count [Pritab Pointer [SYMBOL]] := Error Count [Pritab Pointer
                                                    [SYMBOL]] + 1;
          ERROR := true;
          go to NYD;
CLAR28:   ERMESS (ERR28);
          go to CLAR27P2;
DRET: end DECLARE;

Procedure FOXL;
Begin Integer I, J;
        KKK := 0
        J := 1
        for I := LATAB step 1 until LATAE do
        begin TTT [J] := NAME [I];
            if FL ≠ FLT [I] then go to FOXL3;
            if STAT [I] = 0   STAT [I] = 3 then go to FOXL3;
            if STAT [I] = 1 then begin FLT [I] := FLT [I] - 1;
              go to FOXL3 end then;
FOXL]:  STAT [I] := 3;
            if SWBT [I] = 0 then go to FOXL3 else
            KKK := J; J := J+1;
FOXL3:  end for;
```

```
FOXL4: if KKK = 0 then go to FOXL6;
        ERMESS (ERR24);
       for I = 1 step 1 until KKK do
         ERR26 (TTT [I]);
FOXL5:   ERMESS (ERR25);
FOXL6:   FL := FL - 1;
      end FOXL;

Procedure Advnce; Begin
```

Comment this procedure is similar to SKIP, but does not transmit the symbol currently under scan for translation. In practice, this was accomplished by modifying SKIP, using it, then restoring the modifications;

```
        SKIPO PLUS 7 := SKIPO PLUS 8 := MOD1;
        SKIPO PLUS 7 := MOD2;
        SKIPO PLUS 8 := MOD3
end Advnce;
```

ACKNOWLEDGEMENTS

The syntax checker was based on a method devised by Dr. H. H. Bottenbruch who also did much preliminary planning on the project while he was associated with the Oak Ridge National Laboratory. The author wishes to thank him for his advice and assistance during the early stages of the development of this program.

Thanks are also extended to Drs. A. S. Householder and A. A. Grau for their continued advice, encouragement and support in this project.

ORNL-3399
UC-32 — Mathematics and Computers
TID-4500 (18th ed., Rev.)

INTERNAL DISTRIBUTION

1. Biology Library
2-3. Central Research Library
4. Reactor Division Library
5-6. ORNL — Y-12 Technical Library
Document Reference Section
7-26. Laboratory Records Department
27. Laboratory Records, ORNL R.C.
28. P. A. Agron
29. S. E. Atta
30. R. H. Bassel
31. R. E. Biggers
32. H. P. Carter
33. G. C. Chapman
34. R. R. Coveyou
35. M. H. Davis
36. H. J. deBruin
37. A. C. Downing
38. F. F. Dyer
39. M. Feliciano
40. B. Flores
41. T. B. Fowler
42. A. A. Grau
43. D. A. Griffin
44. W. L. Griffith
45. J. Halperin
46. A. S. Householder
47. M. T. Karkrider
48. W. H. Jordan
49. H. W. Joy
50. L. J. King
51. M. O. Labhart
52. C. E. Larson
53. M. E. LaVerne
54. E. Lee
55. H. A. Levy
56. M. H. Lietzke
57-76. M. P. Lietzke
77. M. J. Mader
78. R. V. Miskell
79. E. C. Moncrief
80. J. Olson
81. C. E. Parker
82. I. R. Parsley
83. D. Patterson
84. C. A. Preskitt
85. J. E. Rowe
86-87. O. W. Russ, Jr. (K-25)
88. H. C. Schweinler
89. M. J. Skinner
90. R. W. Stoughton
91. J. A. Swartout
92. J. G. Sullivan
93. C. D. Susano
94. A. M. Weinberg
95. D. B. Trauger
96. C. S. Williams

EXTERNAL DISTRIBUTION

97. R. L. Berggren, D/596-142, Rocketdyne, 6633 Canoga Avenue, Canoga Park, California
98. L. Breed, Stanford Computation Center, Stanford, California
99. D. B. J. Bridges, D/59-24/201, 3251 Hanover Street, Palo Alto, California
100. T. Connor, Atlantic Refining Company, 260 S. Broad Street, Philadelphia, Pennsylvania
101. D. Drake, National Bureau of Standards, Boulder, Colorado
102. D. Eastwood, Bell Telephone Labs, Murray Hill, New Jersey
103. R. Franciotti, IBM Programming Systems, 1271 Avenue of the Americas, New York 20, New York
104. J. C. Gysbers, D/596-142, Rocketdyne, 6633 Canoga Avenue, Canoga Park, California
105. M. I. Halpern, Lockheed Missiles and Space Company, D/59-25/201, 3251 Hanover Street, Palo Alto, California

106. D. Harrison, Atlantic Refining Company, 260 S. Broad Street, Philadelphia, Pennsylvania
107. W. P. Heising, IBM Programming Systems, 1271 Avenue of the Americas, New York 20, New York
108. L. Hendricks, Mathematical Analysis, D/5426, Sandia Corporation, Albuquerque, New Mexico
109. C. Hipkins, Computer Applications, Inc., c/o Institute for Space Studies, NASA, 475 Riverside Drive, New York 27, New York
110. A. Hirsch, General Electric Company, P. O. Box 988, Huntsville, Alabama
111. C. C. Kunkel, D/592 Data Processing, Rocketdyne, 6633 Canoga Avenue, Canoga Park, California
112. G. Mealy, The Rand Corporation, 1700 Main Street, Santa Monica, California
113. E. G. Richter, Armour Research Foundation, 10 W. 35th Street, Chicago 16, Illinois
114. F. Rotar, Martin Marietta, Box 179 Mail No. A213, Denver, Colorado
115. B. D. Rudin, Lockheed Missiles and Space Company, D/59-25/201, 3251 Hanover Street, Palo Alto, California
116. G. Ryckman, Special Problems Department, General Motors Reseach Laboratory, General Motors Technical Center, Warren, Michigan
117. E. J. Schubert, Beckman Systems Division, 2400 Harbor Blvd., Fullerton, California
118. R. G. Selfridge, Walker Hall, University of Florida, Gainesville, Florida
119. J. N. Snyder, Digital Computer Laboratory, University of Illinois, Urbana, Illinois
120. J. Spall, Room 26-153, Computation Center, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, Massachusetts
121. M. Troost, General Atomic Division, General Dynamics Company, P. O. Box 608, San Diego 12, California
122. J. R. Whitney, D/596-142, Rocketdyne, 6633 Canoga Avenue, Canoga Park, California
123. G. Wiederhold, Computer Center, University of California, Berkeley 4, California
124. V. J. Zapotocky, McDonnell Automation Center, Department 73, Box 516, St. Louis 66, Missouri
125. Research and Development Division, AEC, ORO
126-734. Given distribution as shown in TID-4500 (18th ed., Rev.) under Mathematics and Computers category (75 copies – OTS)