CC-246

COMPUTATION CENTER
Massachusetts Institute of Technology
Cambridge 39, Massachusetts

March, 1965

TO: COMIT and FORTRAN Programmers

FROM: D. C. Matiatos, Department of Chemical Engineering

SUBJECT: COMIT Subroutines for Generating FORTRAN Programs

INTRODUCTION

The purpose of this memo is to describe certain closed COMIT (7,8) subroutines which are helpful for generating FORTRAN (1,3) programs by means of COMIT programs. This type of programming is useful, since it allows one to obtain from the mathematical formulation of a problem, e.g. the differential equations describing it, a FORTRAN program capable of producing numerical solutions of that problem. Such a programming approach is presently being used in the development of a COMIT program for the automatic numerical solution of a class of non-linear partial differential equations associated with diffusion and chemical reaction (4,5,6). It is in connection with this work that the subroutines described in this memo were developed.

The present memo contains information on subroutines COUNT and STATN, which permit the sequential numbering of FORTRAN statements, the FOR-IO package of subroutines, which deals with the input and output of the data of a FORTRAN program, and subroutine LL, which allows the automatic generation of continuation cards whenever a FORTRAN record produced by the COMIT program exceeds 72 characters. The tasks these subroutines can do are quite simple when performed directly by a human programmer. Nevertheless, they do require an appreciable amount of programming effort when performed indirectly by means of another program.

It is assumed throughout this memo that the reader is fairly familiar with both COMIT and FORTRAN. Listings of the various subroutines preceded by a discussion intended to facilitate their use may be found in the Appendix.

ACKNOWLEDGEMENT.

This work was done in part at the Computation Center of the Massachusetts Institute of Technology. The author is grateful for his use of the Center's facilities. He also wishes to express his gratitude to Dr. V. H. Yngve who supervised this project.

NUMBERING OF FORTRAN STATEMENTS: THE COUNT AND STATN SUBROUTINES

FORTRAN statements are labeled by means of an integer (which appears in columns 1-5) so that they can be referred to by means of this number in other statements. If a FORTRAN program is to be generated by a COMIT program, the latter should be capable of producing integers sequentially, one of which is used every time the need for numbering a statement arises. This can be accomplished by means of subroutines COUNT and STATN. The main program transfers to COUNT after introducing two return constituents in shelf 20 so that control will pass directly from COUNT to STATN. Upon return to the main program, the workspace will contain a return constituent followed by an integer, a number of blanks and the contents which the workspace had when COUNT was entered. The digits of the integer and the blanks always add up to six characters (both these strings are fully expanded) and, therefore, they can be used as columns 1-6 of a FORTRAN statement. If, before calling COUNT, the workspace contained at its beginning the remaining columns of the statement, upon return to the main program it would contain a complete numbered FORTRAN statement. To initialize COUNT the main program has to introduce into shelf 122 a fully expanded integer followed by the two marker constituents *N and *P. Every time COUNT is called it increments the integer it finds in shelf 122 by one. It uses the incremented number for labeling a FORTRAN statement and it also replaces by it the number in shelf 122. Thus COUNT need be initialized only once and from then on it will keep numbering statements sequentially.

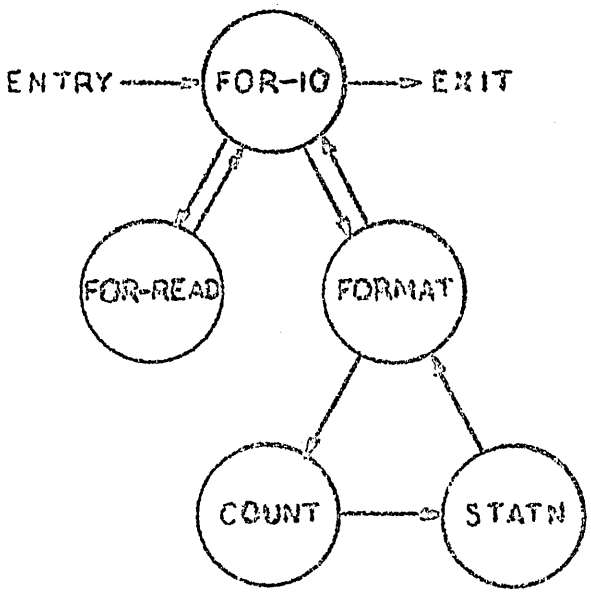
Once a FORTRAN statement has been numbered, the main COMIT program can copy its number and insert it in another statement which refers to the statement numbered by COUNT and STATN. An example of the use of these two subroutines can be found in the next section, which deals with the FOR-IO package. While using COUNT and STATN for numbering statements a programmer ought to be careful that these routines produce numbers which are always less than 32,768. This is a restriction imposed by the FORTRAN system. He should also initialize COUNT in a way that will guarantee that statements numbered by it do not have numbers which coincide with those of statements labeled directly by the main program. For example, in the case of the FOR-IO package (see the next section) COUNT should not assign numbers 1 and 2 to any of the FORMAT statements it numbers, since these two labels are reserved for two FORMAT statements produced by the main program itself.

READING AND PRINTING OF DATA: THE FOR-IO PACKAGE

Most FORTRAN programs are written in a way that enables them to read in data before they perform any actual

calculations. Thus they can be run again and again with different sets of data. In addition many programmers have the FORTRAN program print out the numbers it just read so as to have the output documented by the values of the constant parameters that define a particular calculation. This is also convenient for detecting certain types of errors. If a FORTRAN program is to be generated automatically by means of a COMIT program, the latter should be able to incorporate into the former input-output statements performing the tasks mentioned above. For example, if a COMIT program whose input is a system of partial differential equations is to produce a FORTRAN program for solving them numerically, as is the case in (4,5,6), the COMIT program will first have to scan the differential equations for names of floating and fixed point (in the FORTRAN sense) variables. Once these two lists have been compiled, the COMIT program has to generate FORTRAN statements for reading in and printing out these constant parameters. This task can be accomplished by means of the FOR-10 package.

The FOR-10 package consists of five closed COMIT subroutines arranged as follows:



FOR-10 can be called after any number of legal FORTRAN floating variable names have been introduced in shelf 1 (in compressed form), fixed ones in shelf 2, and COUNT has been properly initialized as described in the preceding section. For example, if the FOR-10 package finds in shelf 1: ALPHA*BETA*EPSILN*R*G*H*FR*ONE, in shelf 2: I*J*K*LAMDA*IOTA*KA*LALPHA*NAB and in shelf 122: *4**N**P, upon return to the main program the workspace will contain a set of expanded FORTRAN statements which are shown enclosed in the box below (this is actually their form after being printed in COMIT format A):

```
1       FORMAT(4E18.8)
2       FORMAT(4I18)
        READ 1, ALPHA, BETA, EPSILN, R
        PRINT 5
5       FORMAT(13X5HALPHA,14X4HBETA,12X6HEPSILN,17X1HR)
        PRINT 1, ALPHA, BETA, EPSILN, R
        READ 1, G, H, FR, ONE
        PRINT 6
6       FORMAT(17X1HG,17X1HH,16X2HFR,15X3HONE)
        PRINT 1, G, H, FR, ONE
        READ 2, I, J, K, LAMDA
        PRINT 7
7       FORMAT(17X1HI,17X1HJ,17X1HK,13X5HLAMBDA)
        PRINT 2, I, J, K, LAMDA
        READ 2, IOTA, KA, LALPHA, NAB
        PRINT 8
8       FORMAT(14XHIOTA,16X2HKA,12X6HLAPHA,15X3HNAB)
        PRINT 2, IOTA, KA, LALPHA, NAB
        CALL EXIT
        END
```

The first statement reads in the first four floating variables in FORMAT 1. The next two statements print out labels for each of these four variables. The fourth statement prints out the four variables themselves. This cycle is repeated until the list of floating variables is exhausted. Then the fixed point variables are taken care of by means of similar sets of statements except that FORMAT 2 is used. The last cycle corresponding to each list may read in four, three, two or one variable depending on the number of them contained in the list. FOR-IO functions properly even when one or both lists are empty. FORMATS 1 and 2 which are used by some of the statements generated by FOR-IO are shown right on top of the box. They are considered as being among the simplest FORTRAN FORMATS (2). The COMIT program that uses FOR-IO should introduce them into the FORTRAN program. Both statements can always be introduced even though one or both variable lists might be empty, since an unused FORMAT statement is not considered an error by FORTRAN. It should also be noted that the longest statement produced by FOR-IO is only 62 characters long including blanks. Thus every statement generated by FOR-IO will fit on one card. By adding the two statements shown below the box one has a complete FORTRAN program. If this is executed with the following data:

```
 0.10000000E 01   0.20000000E 01   0.30000000E 01   0.40000000E 01
-0.10000000E-01  -0.20000000E-01  -0.30000000E-01  -0.40000000E-01
              1                2                3                4
             -1               -2               -3               -4
```

the output obtained is:

ALPHA	BETA	EPSILN	R
.10000000E 01	.20000000E 01	.30000000E 01	.40000000E 01
G	H	FR	ONE
-.99999999E-02	-.20000000E-01	-.30000000E-01	-.39999999E-01
I	J	K	LAMDA
1	2	3	4
IOTA	KA	LALPHA	NAB
-1	2	-3	4

It can be seen that the labels are right-justified on top of each arithmetic field.

FOR-IO was programmed with the particular job described above in mind. However, it is possible to modify the statements it produces so that they are useful for other tasks. For example, after destroying the READ statements, the remaining statements generated by FOR-IO could be used for printing out results. Or after more severe modifications one could even obtain statements for the input and output of arrays.

CONTINUATION CARDS: SUBROUTINE LL

It was stated in the description of the FOR-IO package that the FORTRAN statements it produces are always shorter than 72 characters and thus will fit on one card. In general, however, FORTRAN statements produced by a COMIT program need not satisfy this condition. In that case, continuation records are needed and the long statement will have to be accomodated on more than one card. Subroutine LL (for Line Length) takes care of this task automatically.

This will be demonstrated by means of examples. For instance, if LL finds in the workspace:

```
280 R's (as separate constituents) + *.
```

It will produce:

```
                 72R's(as separate constituents)+*.
+-+-+-+-+-+-+*1+66R's(as separate constituents)+*.
+-+-+-+-+-+-+*2+66R's(as separate constituents)+*.
+-+-+-+-+-+-+*3+66R's(as separate constituents)+*.
+-+-+-+-+-+-+*4+10R's(as separate constituents)+*.
```

When printed or punched in COMIT format A this will look like:

```
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     1RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     2RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     3RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     4RRRRRRRRRR
```

If a statement contains more than 666 characters it cannot be handled by means of continuation cards, since FORTRAN allows only nine of them. What should be done in this case is to break the long statement into two or more statements. This task requires a knowledge of the grammar of FORTRAN and is therefore rather difficult to program. What LL does in such a case can be seen from the following example which involves a statement consisting of 1000 R's:

```
RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     1RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     2RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     3RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     4RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     5RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     6RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     7RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     8RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     9RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     ERRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     ERRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     ERRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     ERRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     ERRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
     ERRRR
```

The E's in column 6 indicate to the programmer that he should take care of this situation himself. In view of the improbability of a statement which cannot fit on 10 cards this is not a serious shortcoming of LL. Furthermore, should a subroutine become available for handling such statements it would be possible to have LL treat the program first and then transfer to the other subroutine which would only process statements having E's in column 6. Subroutine LL recognizes comments and treats them differently than regular statements. For example if it finds in the workspace:

C + 849R's (as separate constituents)++.

the result would look like:

```
CRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
C     RRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRRR
```

Continuation punches are not necessary for comment cards and there is no limit on how long a comment can be. The R's in the above examples stand for any legal FORTRAN character or blank. They were used, since is was much easier to generate long strings of R's rather than legal FORTRAN statements in the program for testing LL.

APPENDIX

A listing of all the subroutines described in this memo may be found at the end of this appendix. Before using any of them, the comments that precede it should be carefully studied. Particular attention should be paid to the various sections of COMIT pseudo-storage, i.e., workspace and shelves, that these routines require to be empty. The user of these routines should avoid naming rules in his main program with rule names that appear in the routines he uses. In general, he should be particularly well aquainted with the use of closed subroutines in COMIT (9). He should also bear in mind that error detection or recovery procedures have _not_ been incorporated in these subroutines and consequently very erratic behavior may occur if they are improperly used. Other than that they have been extensively tested and are believed to be free of errors.

```
          (CLOSED SUBROUTINE FOR-IO)
(FINDS A LIST OF COMPRESSED FLOATING VARIABLE NAMES IN SH. 1 AND A )
(LIST OF FIXED ONES IN SH. 2)
(FOR-IO DOES NOT SAVE THESE LISTS)
(THE WS SHOULD BE EMPTY UPON ENTRY)
(USES SH. 3, 4, 5 AND 6 WHICH SHOULD BE EMPTY UPON ENTRY AND ARE )
(ALSO EMPTY UPON RETURN)
(RETURNS WITH A SET OF EXPANDED IO STATEMENTS IN THE WS)
(CALLS SUBROUTINES FOR-READ AND FORMAT)
(FINDS ITS RETURN IN SH. 20)

(PROCESS FLOATING VARIABLES FIRST)
FOR-IO // FOR-IO1 FLO *
(GENERATE READ STATEMENTS OR RETURN)
FOR-IO1 FLO $=A/FOR-IO2//*S20 1,*A1 1,FOR-IO1 FIX FOR-READ
        FIX  =A/1TO2    //*S20 1,*A2 1,FOR-IO1 RET FOR-READ
        RET  =A+A       //*A6 2,*N20 1 $
(CHANGE FORMAT NUMBERS FROM 1 TO 2 AFTER DESTROYING RETURN CONSTITUENT)
1TO2 $1=0 *
1TO2L $11+*1+$+*.=1+*2+3+4//*Q3 1 2 3 4 1TO2L
* //*X3 FORMATCALL
(DESTROY RETURN CONSTITUENT)
FOR-IO2 $1=0 *
(PUT READ STATEMENT IN SH. 3, PRINT STATEMENT IN SH. 4, REMAINING READ)
(STATEMENTS IN SH. 5 AND CALL FORMAT)
FORMATCALL $+R+E+A+D+$+*.+$=1+2+3+4+5+6+7+8+1+PRINT+6+7+A/FORMATCALL1=
+1+2+3+4+5+6+7//*Q3 1 2 3 4 5 6 7,*E10,*Q4 9 10 11 12,*Q5 8,*S20 13 =
FORMAT
(ALL READ STATEMENTS PROCESSED)
*  FOR=IO1
(PUT FOUR IO STATEMENTS IN SH. 6)
FORMATCALL1 $1+$=A+2+A//*A3 1,*A4 3,*Q6 1 2 3,*X5 FORMATCALL
          (CLOSED SUBROUTINE FOR-READ)
(FINDS A LIST OF VARIABLE NAMES IN THE WS)
(PRODUCES EXPANDED READ STATEMENTS WHICH IT LEAVES IN THE WS)
(FOUR VARIABLES ARE READ IN PER STATEMENT AND ACCORDING TO FORMAT 1)
(THE LAST STATEMENT READS IN THE REMAINING VARIABLES)
(USES SH. 10 WHICH SHOULD BE EMPTY UPON ENTRY AND IS ALSO EMPTY UPON)
(RETURN)
(FOR-READ FINDS ITS RETURN IN SH. 20)

(PRODUCE READ STATEMENTS READING FOUR VARIABLES)
FOR-READ $1+$1+$1+$1=-----=-READ-*1,-+1+,-+2+,-+3+,-+4+*.//*Q10 1 2 3 4 =
5 6 7 8 9 FOR-READ
(TAKE CARE OF REMAINING VARIABLES. PUT STATEMENTS IN WS)
* $1+$1+$1=-------READ-*1,-+1+,-+2+,-+3+*.//*Q10 1 2 3 4 5 6 7,*X10 FOR-=
READR
* $1+$1=-------READ-*1,-+1+,-+2+*.//*Q10 1 2 3 4 5,*X10 FOR=READR
* $1=-------READ-*1,-+1+*.//*Q10 1 2 3,*X10 FOR-READR
* //*X10 *
(EXPAND READ STATEMENTS. RETURN)
FOR-READR $=A+1//*E2,*N20 1 $
        (CLOSED SUBROUTINE FORMAT)
(FINDS IN THE WS A READ STATEMENT PRODUCED BY FOR-READ AND PRODUCES)
(A PRINT STATEMENT AND A FORMAT STATEMENT WHICH WILL LABEL THE INPUT)
```

```
(DATA AND LEAVES THEM EXPANDED IN THE WS)
(CALLS SUBROUTINES COUNT AND STATN)
(USES SH. 10 WHICH SHOULD BE EMPTY UPON ENTRY AND IS ALSO EMPTY UPON)
(RETURN)
(FORMAT FINDS ITS RETURN IN SH. 20)

(DESTROY PART OF READ STATEMENT. ADD FINAL COMMA)
(AND MARKER AT BEGINNING OF WS)
FORMAT $13+$**.=*M+2+, *
(PRODUCE X AND H FIELDS FOR THE VARIABLES)
FORMATL *M+-+$1+,=1+*1*7X*1H+3+4//*Q10 2 3 4 FORMATL
*       *M+-+$2+,=1+*1*6X*2H+3+4//*Q10 2 3 4 FORMATL
*       *M+-+$3+,=1+*1*5X*3H+3+4//*Q10 2 3 4 FORMATL
*       *M+-+$4+,=1+*1*4X*4H+3+4//*Q10 2 3 4 FORMATL
*       *M+-+$5+,=1+*1*3X*5H+3+4//*Q10 2 3 4 FORMATL
*       *M+-+$6+,=1+*1*2X*6H+3+4//*Q10 2 3 4 FORMATL
(ALL VARIABLES PROCESSED. PUT X AND H FIELDS IN WS)
(PUT MARKER AT END OF WS)
* *M=A+1//*A10 1 *
(REMOVE FINAL COMMA AND MARKER. COMPLETE AND NUMBER FORMAT STATEMENT)
* $+,+*M=A/FORMATP+A/STATN+FORMAT*(+1+*)*.//*S20 1 2 COUNT
( PRODUCE PRINT STATEMENT. RETURN)
FORMATP $1+$+-+$=2+2+3+4//*Q10 1,*E4,*X10 *
* $=A+-------PRINT-+1+*.+A//*E2,*A10 5,*N20 1 $
                (CLOSED SUBROUTINE COUNT)
(A VARIATION OF A SIMILAR PROGRAM OBTAINED FROM DR. YNGVE)
(IT IS INITIALIZED BY INTRODUCING A NUMBER FOLLOWED BY THE MARKERS -
*N AND *P INTO SHELF 122. THIS STRING SHOULD BE THE ONLY CONTENTS OF-
 SHELF 122 UPON ENTRY AND IT SHOULD BE FULLY EXPANDED.)
(UPON EXIT FROM COUNT THE W.S. CONTAINS THE NUMBER THAT WAS PLACED IN -
SH. 122 INCREMENTED BY ONE FOLLOWED BY THE TWO MARKERS *N AND *P. -
THIS STRING IS FULLY EXPANDED. IT IS FOLLOWED BY THE ORIGINAL CONTENTS -
OF THE W.S.)
(UPON EXIT SH. 122 ALSO CONTAINS THE STRING THAT WAS ADDED BY COUNT -
TO THE BEGINNING OF THE W.S. THUS FOR CONSECUTIVE NUMBER GENERATION -
COUNT IS SELF-INITIALIZING)
(UNLIKE DR. YNGVE'S VERSION INITIAL ZEROES ARE NOT NECESSARY)
      (COUNT FINDS ITS RETURN IN SH. 20)

(PUT NUMBER AND MARKERS AT THE BEGINNING OF THE W.S.)
COUNT $=A+1//*A122 1 *
(IF DIGIT BEFORE *N IS NOT 9, INCREMENT IT BY ONE)
COUNT1 *0+*N=*1+2 PC
*      *1+*N=*2+2 PC
*      *2+*N=*3+2 PC
*      *3+*N=*4+2 PC
*      *4+*N=*5+2 PC
*      *5+*N=*6+2 PC
*      *6+*N=*7+2 PC
*      *7+*N=*8+2 PC
*      *8+*N=*9+2 PC
(IF IT IS 9, REPLACE IT BY A ZERO AFTER THE *N AND HANDLE CARRY BY -
GOING TO COUNT1)
*      *9+*N=2+*0 COUNT1
(NO DIGIT TO ADD CARRY TO. ASSUME AN IMPLIED INITIAL ZERO)
```

```
* *N=*1+1 *
(PUT ANY ZEROES AFTER *N IN FRONT OF IT)
PC *N+$+*P=2+1+3 *
(LEAVE NEW NUMBER AND MARKERS AT THE BEGINNING OF W.S. AND IN SH. 122. -
RETURN)
* $+*N+*P=1+2+3+1+2+3//*Q122 1 2 3,*N20 1 $
            (CLOSED SUBROUTINE STATN)
(TO NUMBER A FORTRAN STATEMENT ONE SHOULD CALL COUNT LEAVING COLUMNS)
(7-END OF STATEMENT IN THE WS)
(THEN CONTROL SHOULD BE TRANSFERRED DIRECTLY FROM THE RETURN OF COUNT)
(TO STATN WHICH WILL PRODUCE A STATEMENT HAVING A LEGAL COL.1-6 FIELD)
(AND LEAVE IT IN THE WS)
(STATN MUST BE NORMALLY CALLED BY A $.UPON ENTRY IT DESTROYS THE FIRST)
(CONSTITUENT OF THE WS)
(STATN FINDS ITS RETURN IN SH. 20)

(DESTROY RETURN CONSTITUENT. ADD 5 BLANKS AFTER NUMBER.)
STATN $1+$+*N+*P=2+-----+3+4//*E2 *
(REMOVE EXCESS BLANKS AND MARKERS. RETURN.)
* $6+$+*N+*P=A+1//*N20 1 $
                (CLOSED SUBROUTINE LL)
(FINDS AN EXPANDED FORTRAN STATEMENT OR STATEMENTS IN THE W.S. WHICH)
(SHOULD NOT CONTAIN ANYTHING ELSE)
(PRODUCES SEQUENTIALLY NUMBERED CONTINUATION CARDS FOR STATEMENTS WHICH)
(ARE LONGER THAN 72 CHARACTERS)
(IF A STATEMENT CANNOT BE ACCOMODATED ON 10 CARDS, LL WILL PRODUCE)
(FURTHER CONTINUATION CARDS HAVING AN E IN COLUMN 6. WHENEVER THIS)
(CONDITION OCCURS, THE USER MUST DEFINE A PART OR PARTS OF THE)
(EXCESSIVELY LONG STATEMENT BY MEANS OF OTHER STATEMENTS, SO AS TO)
(SHORTEN IT TO A LEGAL LENGTH)
(COMMENTS ARE RECOGNIZED. CONTINUATION CARDS FOR THEM HAVE A C IN)
(COLUMN 1 FOLLOWED BY 5 BLANKS AND THEN TEXT UP TO AND INCLUDING)
(COLUMN 72)
(LL RETURNS WITH THE PROCESSED, EXPANDED FORTRAN PROGRAM OR STATEMENT)
(IN THE W.S. READY TO BE PUNCHED ON CARDS. FOR PRINTING A BLANK SHOULD)
(BE INTRODUCED BEFORE EACH RECORD BY MEANS OF A RULE OF THE TYPE)
( A $+*.=-+1+2//*WAM1 2 3 A )
(LL USES SHELVES 1 AND 2 WHICH SHOULD BE EMPTY UPON ENTRY AND ARE ALSO)
(EMPTY UPON RETURN)
(LL FINDS ITS RETURN IN SH. 20)

(LEAVE ONLY FIRST STATEMENT IN W.S.. PLACE MARKER BEFORE IT)
LL $+*.+$=*M+1+2+3//*Q1 4 LL0
(PROGRAM PROCESSED. PUT IT IN W.S., EXPAND IT AND RETURN)
* $=A+A//*N20 1,*A2 2,*E2 5
(IS THIS A COMMENT)
LL0 *M+C=2 LLC
(NO)
* *M=0 LL1
(YES)
LLC $72+$1+$+*.=1+*.C-----=+2+3+4//*Q2 1 2 LLCL
* $=//*Q2 1,*X1 LL
LLCL $66+$1+$+*.=1+*.C-----=+2+3+4//*Q2 1 2 LLCL
* $=//*Q2 1,*X1 LL
(DOES STATEMENT FIT ON ONE CARD)
```

```
LL1 $72+$1+$+*.=1+*.=====#1+2+3+4//*Q2 1 2 LL2
(YES)
* $=//*Q2 1,*X1 LL
(NO. DOES IT FIT ON TWO CARDS)
LL2 $66+$1+$+*.=1+*.=====#2+2+3+4//*Q2 1 2 LL3
(YES)
* $=//*Q2 1,*X1 LL
(NO. DOES IT FIT ON THREE CARDS)
LL3 $66+$1+$+*.=1+*.=====#3+2+3+4//*Q2 1 2 LL4
* $=//*Q2 1,*X1 LL
LL4 $66+$1+$+*.=1+*.=====#4+2+3+4//*Q2 1 2 LL5
* $=//*Q2 1,*X1 LL
LL5 $66+$1+$+*.=1+*.=====#5+2+3+4//*Q2 1 2 LL6
* $=//*Q2 1,*X1 LL
LL6 $66+$1+$+*.=1+*.=====#6+2+3+4//*Q2 1 2 LL7
* $=//*Q2 1,*X1 LL
LL7 $66+$1+$+*.=1+*.=====#7+2+3+4//*Q2 1 2 LL8
* $=//*Q2 1,*X1 LL
LL8 $66+$1+$+*.=1+*.=====#8+2+3+4//*Q2 1 2 LL9
* $=//*Q2 1,*X1 LL
LL9 $66+$1+$+*.=1+*.=====#9+2+3+4//*Q2 1 2 LLE
* $=//*Q2 1,*X1 LL
(DOES IT FIT ON 10 CARDS)
LLE $66+$1+$+*.=1+*.=====E +2+3+4//*Q2 1 2 LLE
* $=//*Q2 1,*X1 LL
```

Literature Cited

1. Corbató, F. J., "An Abbreviated Description of the FORTRAN Compiler Language", M.I.T., Computation Center Memo CC-164-1, Cambridge 39, Mass., 1960.

2. Ibid., page 6.

3. I.B.M., "Reference Manual 709/7090 FORTRAN Programming System", Form C28-6054-2, New York, 1961.

4. Matlatos, D. C., "The Automatic Solution of Numerical Problems with a Digital Computer", 6.541J Project Report, M.I.T., Cambridge 39, Mass., 1963. (Available at the M.I.T. Computation Center Document Room).

5. Matlatos, D. C., "A New Automatic Compiler for the Numerical Solution of Diffusion Equations", 6.681 Project Report, M.I.T., Cambridge 39, Mass., 1964. (Available at the M.I.T. Computation Center Document Room).

6. Matlatos, D. C., "Further Work on the Automatic Compiler for Diffusion Equations", 6.682 Project Report, M.I.T., Cambridge 39, Mass., 1964. (Available at the M.I.T. Computation Center Document Room).

7. Yngve, V. H., et al., "COMIT Programmers' Reference Manual", The M.I.T. Press, Cambridge 39, Mass., 1962.

8. Yngve, V. H., et al., "An Introduction to COMIT Programming", The M.I.T. Press, Cambridge 39, Mass., 1962.

9. Ibid., pp. 45-47.