A FORTRAN STATEMENT ANALYZER WRITTEN IN PL/1

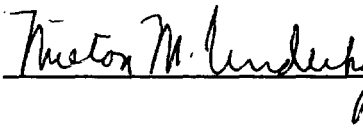
AN HONORS THESIS (ID 499)

BY

DEBRA J. STEELE

THESIS DIRECTOR

DR. MILTON UNDERKOFFLER


(ADVISOR'S SIGNATURE)

BALL STATE UNIVERSITY

MUNCIE, INDIANA

SPRING, 1974

INTRODUCTION

This program is a Fortran IV statement analyzer written in PL/1 for the IBM-360 Computer at Ball State University. It deals with only the most basic Fortran program, handling only the READ, WRITE, FORMAT, STOP, END, and ARITHMETIC statements. Any other statements allowed in the Fortran IV language are considered invalid. They will not be recognized by this program and will be flagged with various error messages. Any particular feature of Fortran that is associated with these statements, such as arrays with the DIMENSION statement, are also considered as errors.

Each statement is read in character form one at a time into the array CC. This array is a single dimension array of length 80 with each level capable of containing one character. This conforms to the columns on the punched card and is used for reference in checking the various statements.

There are several restrictions placed on the Fortran coding. Some are related only to a particular statement and will be explained within the description of the handling of that statement. Others are general and are explained below. Many of these restrictions are the same as the Fortran compiler places on a program while others are particular restrictions of this analyzer.

I. GENERAL RESTRICTIONS OF THE FORTRAN CODING

As with the Fortran compiler, columns one through five (1-5) are for the statement numbers. Only numeric characters may appear in these columns, and any other character in columns one through five (1-5) will cause the statement to be flagged with ****** ERROR ****** - INVALID STATEMENT NUMBER: STATEMENT NUMBER APPEARS MORE THAN ONCE IN PROGRAM.

A character in column six (6) denotes a continuation card. Although this program does not handle continuation cards, it will flag a statement with a character in column six (6) if there are numbers preceding it. The error message is ****** ERROR ****** - NUMBERS PRECEDE CONTINUATION CARD.

Each Fortran statement must begin in column seven (7). After column seven, blanks may appear anywhere it is logical, i.e. between words, before and after arithmetic signs, etc., unless otherwise specified below. The blanks are skipped and do not process through the analyzer.

Any combination of letters may be used as variable names. However, these names may only be six (6) letters long. If the name is longer than six (6) letters the statement will be flagged.

Only columns seven through seventy-two (7-72), inclusive, may be used for the Fortran statement. Anything beyond column seventy-two (columns 73-80) are ignored. However, if a character does appear in columns seventy-three through eighty (73-80), the warning, ****** WARNING ****** - COLUMNS 73-80 NOT BLANK, will be printed.

II. STATEMENT NUMBERS

As stated previously, the statement numbers appear in columns one through five (1-5). Any integer number from 0 to 99999, inclusive, is acceptable. The numbers are stored in the array NUM which is dimensioned (25,5). This allows twenty-five (25) different statement numbers to appear in the program, which, of course, can be increased or decreased by changing the dimension of the array.

Once a number is stored, it is checked to see if it is already in the array NUM. If it does appear in NUM, an error message, ****** ERROR ******

- INVALID STATEMENT NUMBER: STATEMENT NUMBER APPEARS MORE THAN ONCE IN PROGRAM, is printed. The repetitive number is then blanked out of NUM.

After the statement number has been checked, column seven (7) is compared with the first letter of each of the key words, R for READ, F for FORMAT, W for WRITE, S for STOP, and E for END. If one of these letters is in column seven (7), control is transferred to the respective division the letter represents. Otherwise, the statement is assumed to be an arithmetic assignment statement and is processed through the arithmetic statement division.

Each division first checks to see if there is an equal sign on the card (columns 7 through 72). If no equal sign appears, the statement continues processing through its division. If an equal sign does appear, it is assumed the statement is an arithmetic assignment and control it transferred to the arithmetic statement division.

The next check in each division is to see that the key word is spelled correctly. This is done by concatenating columns seven (7) through n, where n is six plus the number of letters in the key word, e.g. The columns concatenated for the READ statement would be columns seven (7) through ten (10), inclusive. The concatenated string is then compared with the key word string which was read into the array KE1 at the beginning of the program. If the strings match, processing continues through that division. If the strings do not match, the error message, ****** ERROR ****** - UNDECODABLE STATEMENT, is printed, and the next statement is read.

III. THE READ STATEMENT

This program places certain additional restrictions on the coding of

the READ statement. As with all other statements, the coding must begin in column seven (7). There cannot be a space between the word READ and the parenthesis containing the number for the read unit and the format statement number, nor can there be blanks within the parenthesis, i.e. Columns seven (7) through ten (10) must contain the word READ. Column eleven (11) must contain a left parenthesis or the error message, ****** ERROR ****** - MISSING LEFT PARENTHESIS, is printed. Column twelve (12) must contain a five (5) since that is the code for the reader at Ball State University, or the statement is flagged with ****** ERROR ****** - READ ON WRONG UNIT. Column thirteen (13) must contain a comma, or the statement is flagged with ****** ERROR ****** - MISSING COMMA.

The processing in the read division begins with storing the statement number in the array STNO. If the number is greater than 99999 or longer than five (5) characters, the statement is flagged with ****** ERROR ****** - STATEMENT NUMBER TOO LONG. The numbers in STNO are checked at the end of the program with the numbers stored in NUM, the array containing all the statement numbers in the program. If one of the numbers in STNO is not in NUM, the error message, ****** ERROR ****** - MISSING STATEMENT NUMBER _____, is printed.

The processing continues by checking for a left parenthesis. If one other than the one containing the read unit and format statement number appear, the error message, ****** ERROR ****** - INVALID LEFT PARENTHESIS, is printed. Since no arrays are handled by this program, the appearance of a left parenthesis other than noted above is invalid.

The next step is to check for a right parenthesis. When a right parenthesis is encountered, a counter is incremented by one. At the end of

the READ division this counter is tested. If its value is less than one (1), the error message, ****** ERROR ****** - MISSING RIGHT PARENTHESIS, is printed as at least one right parenthesis must appear to close the read unit and format statement group. If the counter is greater than one (1), the error message, ****** ERROR ****** - INVALID RIGHT PARENTHESIS, is printed as only one right parenthesis should appear because arrays are not handled by this program. The same error message is printed if the right parenthesis is not preceded by a number.

If a letter is encountered, it is checked to determine if it represents a fixed point or floating point variable. If the first letter is any of the group of letters A to H or O to Z, it represents a floating point variable. If it is from the group of letters I to N, it represents a fixed point variable. If a letter is the first letter of a variable name, an "I" for fixed point or an "F" for floating point is stored in the array R for reference with the format statement. The letter is then stored in the respective array representing fixed point and floating point variables, FVAR for fixed point variables and PVAR for floating point variables. If the letter is not the first letter of a variable name, it is immediately stored in the proper array.

If the variable name is longer than six (6) letters, the error message, ****** ERROR ****** - VARIABLE NAME TOO LONG: TRUNCATED TO 6 LETTERS, is printed. Only the first six (6) letters of the variable name are stored in the respective array. After all 72 columns have been checked, the next statement is processed.

IV. THE WRITE STATEMENT

As with the READ statement, the WRITE statement restricts the use of blanks after the word WRITE and within the parenthesis containing the unit for printing and the format statement number. Columns seven through eleven (7-11) must contain the word WRITE. Column twelve (12) must contain a left parenthesis, or the statement is flagged with ****** ERROR ****** - MISSING LEFT PARENTHESIS. Column thirteen (13) must contain a six (6), the code for the printer at Ball State University. If a six (6) does not appear in column thirteen (13) the error message ****** ERROR ****** - WRITE ON WRONG UNIT, is printed. A comma must appear in column fourteen (14), or the statement is flagged with ****** ERROR ****** - MISSING COMMA.

After the preceding columns have been checked, the processing continues by storing the statement number of the format for the WRITE statement in the array WSTNO. If this number is greater than 99999 or longer than 5 characters, the statement is flagged with ****** ERROR ****** - STATEMENT NUMBER TOO LONG. At the end of the program, the numbers in WSTNO are compared with the numbers in NUM. If a number is in WSTNO, but not in NUM, the error message, ****** ERROR ****** - MISSING STATEMENT NUMBER _____, is printed.

The processing continues by checking for a left parenthesis. If one appears, the statement is flagged with ****** ERROR ****** - INVALID LEFT PARENTHESIS because arrays are not dealt with this program, and, therefore, a left parenthesis is invalid.

The next step is checking for right parentheses. If one is found a counter, which is initialized to zero each time through the WRITE division, is incremented by one. If this counter is less than one at the conclusion

of the processing for that statement, the error message, ****** ERROR ****** - MISSING RIGHT PARENTHESIS, is printed. If the counter is greater than one, the statement is flagged with ****** ERROR ****** - INVALID RIGHT PARENTHESIS. If the right parenthesis is not preceded by a number, the same error message is printed.

The processing continues by checking for letters. A letter is checked for the type of variable it represents, fixed point or floating point. If it is the first letter of a variable, an "I" or an "F" is stored in the array R for reference to the format statement. The letter is then stored in a temporary storage area until the variable name is completely stored. If the letter is not the first letter of a variable name, it is immediately stored in the proper temporary storage area. If the variable name is longer than six (6) letters, the error message, ****** ERROR ****** - VARIABLE NAME TOO LONG: TRUNCATED TO 6 LETTERS, is printed. Only the first six (6) letters of the variable are stored in the temporary storage area.

Once the variable name is completely stored, it is then compared to the variables stored in the respective variable array, i.e., either with the variable names in FVAR if it represents a fixed point variable or with the variable names in PVAR if it represents a floating point variable. If it does not match any of the variable names in the respective array, the error message, ****** ERROR ****** - VARIABLE UNDEFINED, is printed. After all seventy-two columns have been checked, the next statement is read and processed.

V. THE FORMAT STATEMENT

The restrictions placed on the FORMAT statement are more numerous than those placed on the READ or WRITE statement. First, the FORMAT must follow the READ or WRITE it refers to. This is necessary because of the way the Fortran statements are processed. Since each statement is read and processed independently, there is no way to check ahead or back in the Fortran program for FORMAT statements. Second, the left parenthesis after the word FORMAT must be in column thirteen (13). Next, there can be no repetition factors. That is, the following forms of FORMAT statement or any combination of them will be flagged with the error message following them.

```
1     FORMAT(4I2)
****** ERROR ****** - INVALID NUMBER PRECEDING I-FIELD

2     FORMAT(3F4.2)
****** ERROR ****** - INVALID NUMBER PRECEDING F-FIELD
```

Another restriction is that no alphanumeric characters can be written. Since there can be no repetition factors and since no alphanumeric characters or arrays are allowed, there can be no nesting of parenthesis within the FORMAT grouping parenthesis. The error message for a statement of this type will be either ****** ERROR ****** - INVALID LEFT PARENTHESIS or ****** ERROR ****** - INVALID RIGHT PARENTHESIS. Finally, the largest F-field is I999. If the F-field is larger than F99.9 the error messages, ****** ERROR ****** - MISSING . FOLLOWING F-FIELD and ****** ERROR ****** - NUMBER FOLLOWING I- or F-FIELD TOO LONG, will be printed. If the I-field is longer than I999, the statement will be flagged with ****** ERROR ****** - NUMBER FOLLOWING *- or F-FIELD TOO LONG.

The processing in the format division begins by checking for the errors listed above. This is done by checking each column for a number. If a number appears, the columns preceding and following it are checked to

see that characters in them are valid. If not, the proper error message is printed.

The processing continues by checking for an "I" or "F". If one is encountered, it is put into the array R1. This array is compared at the end of the format division with the array R, the array in which the variable types in the READ and WRITE are stored. If R1 and R are not identical, the error message, ****** ERROR ****** - FORMAT CODE AND DATA TYPE DO NOT MATCH, is printed.

The next step is to check for parentheses. If a left parenthesis appears, the statement is flagged with ****** ERROR ****** - INVALID LEFT PARENTHESIS since no parentheses are allowed within the format grouping parentheses. If a right parenthesis is encountered, a counter is incremented by one. At the end of the division this counter is tested. If it is greater than one, the statement is flagged with ****** ERROR ****** - INVALID RIGHT PARENTHESIS. If it is less than one the error message ****** ERROR ****** - MISSING RIGHT PARENTHESIS, is printed. If it equals one, it is assumed this is the closing right parenthesis for the FORMAT grouping. After all seventy-two columns have been checked, the next statement is read.

VI. THE ARITHMETIC STATEMENT

There are no restrictions other than the general restrictions placed on the arithmetic statements. The operations are denoted by +, addition; -, subtraction; *, multiplication; /, division; and ** exponentiation. The order of execution of these operations is exponentiation followed by multiplication and division, and finally addition and subtraction. Parenthesis are allowed to force the order of operation to be different from above.

The processing begins by storing the variable on the left of the equal sign in the proper array, PVAR or FVAR, depending upon the type of variable represented, floating or fixed point. If this variable is longer than six (6) letters, the error message ****** ERROR ****** - NAME TOO LONG: TRUNCATED TO 6 LETTERS, is printed. Only the first six (6) letters are stored in the array.

The next step is to check for an equal sign. If one does not appear before any other character, the error message ****** ERROR ****** - INVALID EXPRESSION ON LEFT OF EQUAL SIGN, is printed.

The other variable names, if any, are checked to see that they have been defined prior to the statement in which they are used. If they do not appear in PVAR or FVAR, the statement is flagged with ****** ERROR ****** - VARIABLE UNDEFINED.

The processing continues by checking for right and left parentheses. If one is found, a separate counter for each is incremented by one. These counters are tested at the end of the division. If they are not equal, the error message, ****** ERROR ****** - UNMATCHED PARENTHESIS, is printed. After all seventy-two columns have been checked, the next statement is read.

VII. THE STOP STATEMENT

There are no restrictions placed on the stop statement other than the ones listed in the general restrictions. When a STOP is encountered, a counter is incremented by one. At the end of the program, this counter is tested. If it is greater than one, the error message, ****** ERROR ****** - MULTIPLE STOP STATEMENTS, is printed, and if it is less than one, the error message, ****** ERROR ****** - MISSING STOP STATEMENT, is printed.

After the counter has been incremented, the next statement is read.

VIII. THE END STATEMENT

As with the STOP statement,there are no restrictions placed on the END statement other than the ones listed in the general restrictions. When an END is found, a counter is incremented by one. At the end of the program this counter is tested. If it is greater than one, the error message ****** ERROR ****** - MULTIPLE END STATEMENTS, is printed. If it is less than one, the error message ****** ERROR ****** - MISSING END STATEMENT, is printed. After the counter has been incremented, the next statement is read.

IX. CONCLUSION

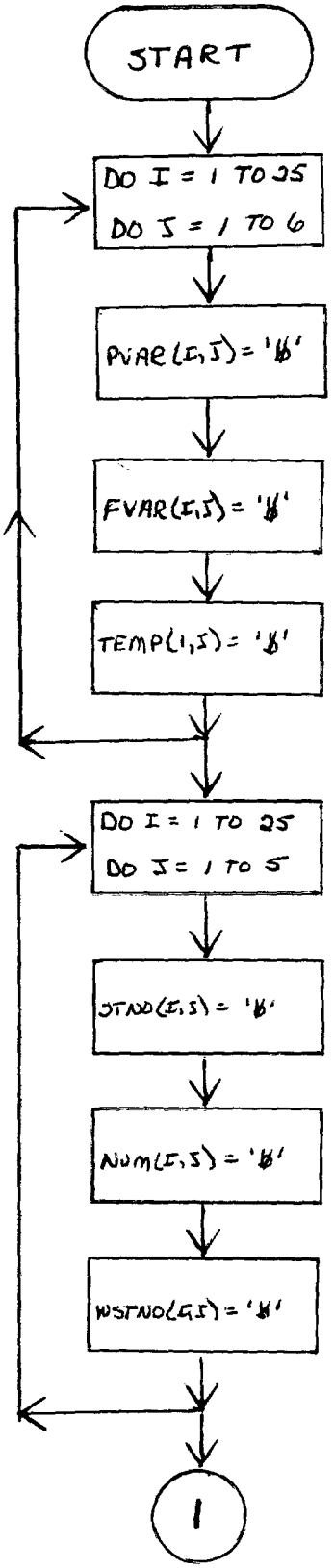
This program offers much room for expansion. The various other types of statements could be incorporate, and subroutines, and functional subroutines as well as other features of the Fortran language could be added. The program could be expanded into a compiler, also, that executes the Fortran program as well.

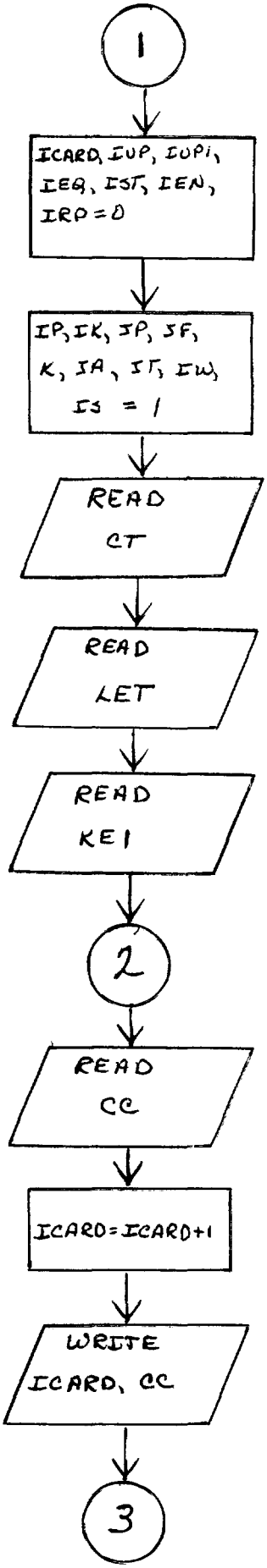
The main disadvantage of this program is the time it takes to execute. This is due to the numerous loops required to check each column on the punched card. Another language such as SNOBOL which is more character-string oriented may prove to be more efficient.

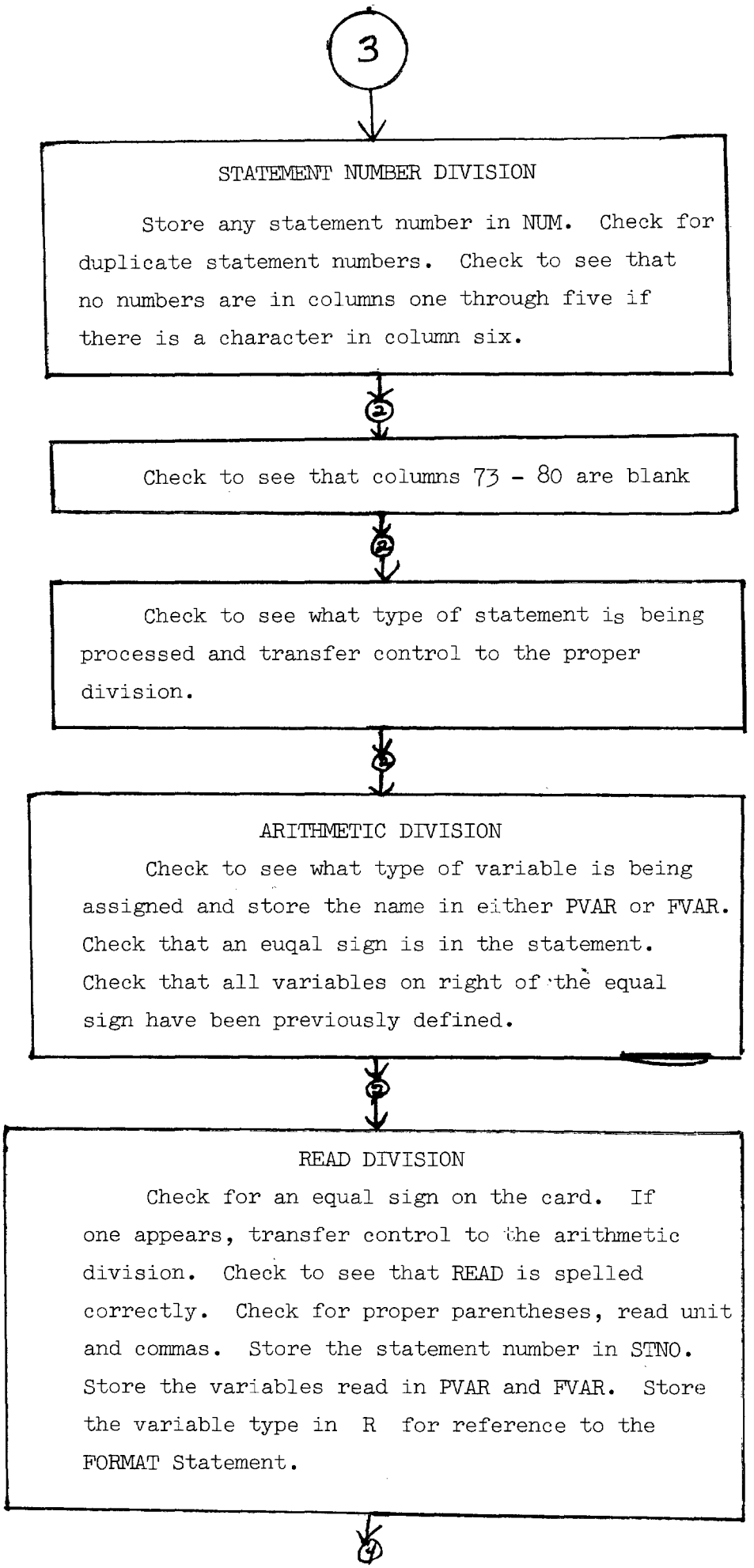
The PL/1 program itself is fairly straight forward. The looping at times becomes difficult to follow but the rest of the program is simple. It was challenging to discover the proper methods to check each column properly, but these problems soon became routine to the program.

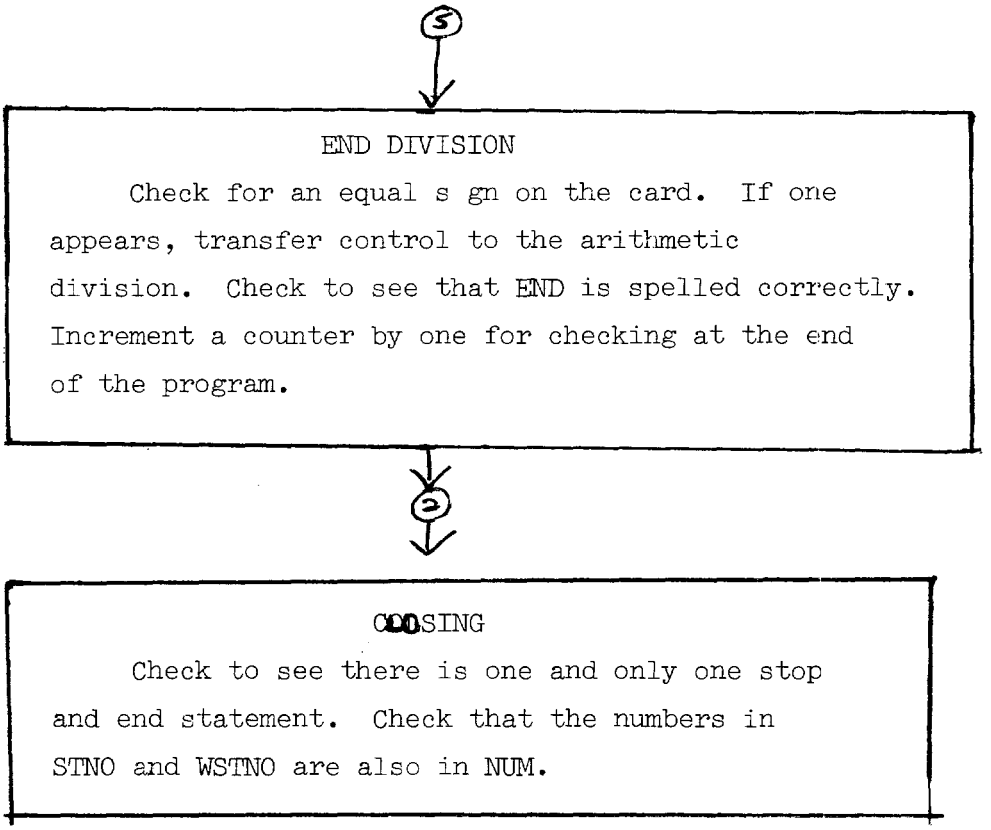
This program written in PL/1 to analyze Fortran IV statements is complete in its handling of the basic READ, WRITE, FORMAT, STOP, END, and arithmetic statements. While most of the checking is for syntax errors, some errors that would occur at execution time, such as undefined variables, are noted. The error messages are concise and clearly indicate the problem in the statement. Any Fortran program, which meets the stated restrictions, that is processed through this PL/1 analyzer will have any errors which occur flagged much as they would by the Fortran compiler.

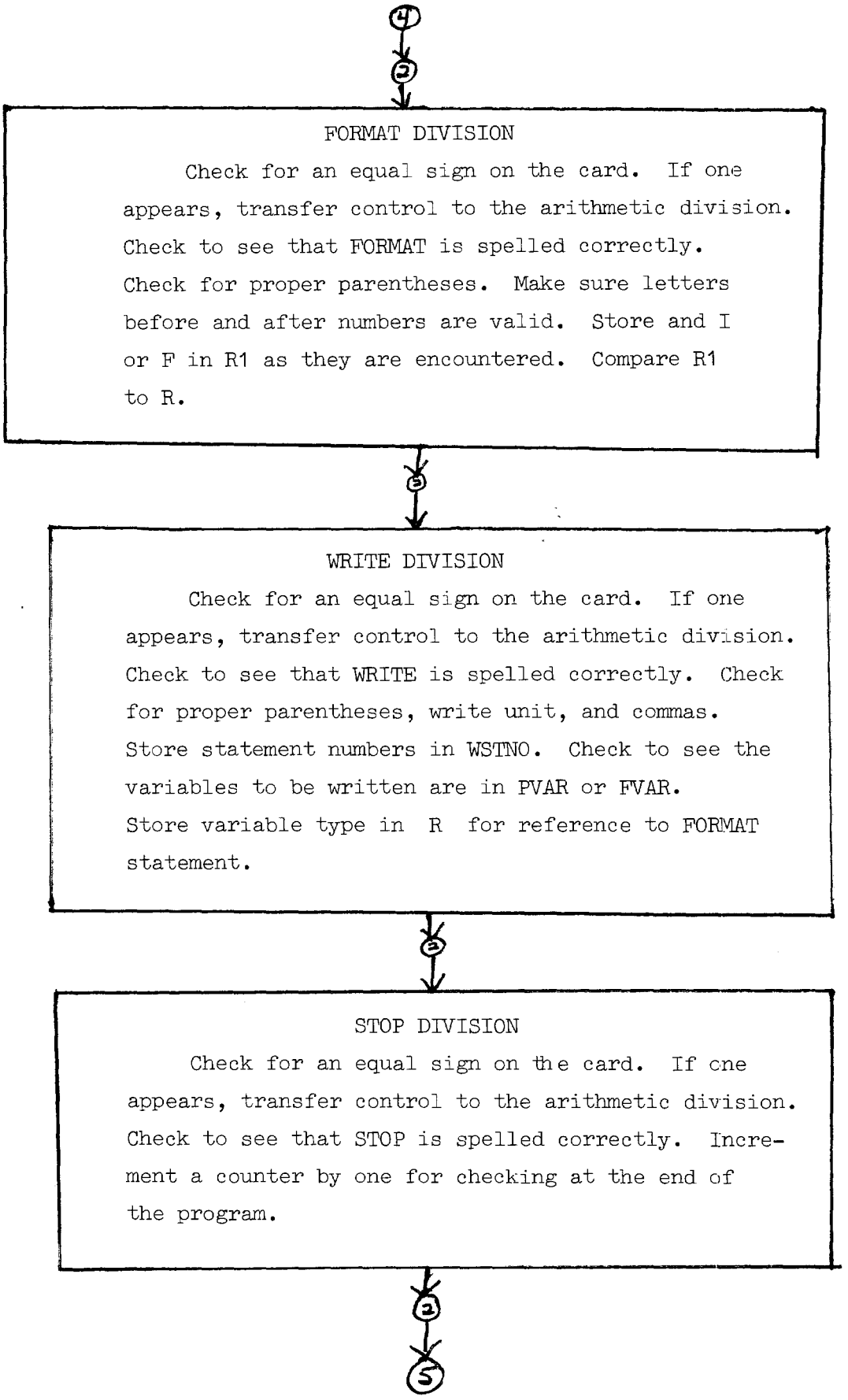
FLOW CHART


START
DO I = 1 TO 25
DO J = 1 TO 6
PVAR(I,J) = 'B'
FVAR(I,J) = 'B'
TEMP(I,J) = 'B'
DO I = 1 TO 25
DO J = 1 TO 5
STNO(I,J) = 'B'
NUM(I,J) = 'B'
WSTNO(I,J) = 'B'
1


1
ICARD, IUP, IUPi,
IEQ, IST, IEN,
IRP = 0
IP, IK, JP, JF,
K, JA, IT, IW,
IS = 1
READ
CT
READ
LET
READ
KEI
2
READ
CC
ICARD = ICARD + 1
WRITE
ICARD, CC
3


3
STATEMENT NUMBER DIVISION
Store any statement number in NUM. Check for duplicate statement numbers. Check to see that no numbers are in columns one through five if there is a character in column six.
1
Check to see that columns 73 - 80 are blank
2
Check to see what type of statement is being processed and transfer control to the proper division.
4
ARITHMETIC DIVISION
Check to see what type of variable is being assigned and store the name in either PVAR or FVAR. Check that an euqal sign is in the statement. Check that all variables on right of the equal sign have been previously defined.
5
READ DIVISION
Check for an equal sign on the card. If one appears, transfer control to the arithmetic division. Check to see that READ is spelled correctly. Check for proper parentheses, read unit and commas. Store the statement number in STNO. Store the variables read in PVAR and FVAR. Store the variable type in R for reference to the FORMAT Statement.
6


5
END DIVISION
Check for an equal s gn on the card. If one appears, transfer control to the arithmetic division. Check to see that END is spelled correctly. Increment a counter by one for checking at the end of the program.
2
CLOSING
Check to see there is one and only one stop and end statement. Check that the numbers in STNO and WSTNO are also in NUM.


4
2
FORMAT DIVISION
Check for an equal sign on the card. If one appears, transfer control to the arithmetic division. Check to see that FORMAT is spelled correctly. Check for proper parentheses. Make sure letters before and after numbers are valid. Store and I or F in R1 as they are encountered. Compare R1 to R.
3
WRITE DIVISION
Check for an equal sign on the card. If one appears, transfer control to the arithmetic division. Check to see that WRITE is spelled correctly. Check for proper parentheses, write unit, and commas. Store statement numbers in WSTNO. Check to see the variables to be written are in PVAR or FVAR. Store variable type in R for reference to FORMAT statement.
2
STOP DIVISION
Check for an equal sign on the card. If one appears, transfer control to the arithmetic division. Check to see that STOP is spelled correctly. Increment a counter by one for checking at the end of the program.
2
5