FORTRAN

What is FORTRAN?

FORTRAN is a general purpose programming language, mainly used for mathematical computation. FORTRAN stands for FORmula TRANslation. The work on FORTRAN started in 1950's at IBM and there have been many versions since. The most common FORTRAN version is still FORTRAN 77.

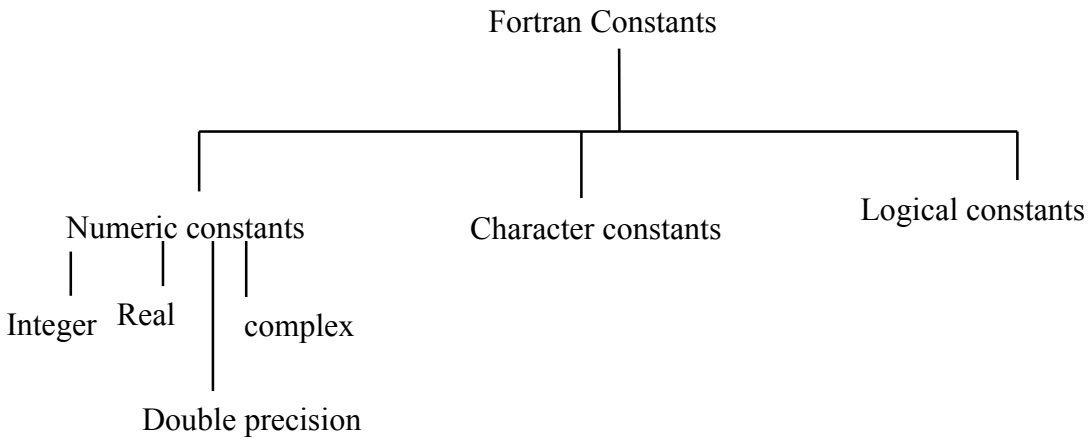
CHARACTER SET:

1. Alphabets: A, B, C, D, …, Z (Upper case)
2. Digits: 0, 1, 2, 3,…,9
3. Special characters: + - * / ( ) , . ' $: = blank space

KEY WORDS:

ASSIGN	CALL	CHARACTER	COMMON
COMPLEX	CONTINUE	DATA	DIMENSION
DO	DOUBLE PRECISION	ELSE	ELSE IF
END	ENDIF	EXTERNAL	FORMAT
FUNCTION	GOTO	IMPLICIT	INTEGER
SUBROUTINE	READ	WRITE	RETURN
OPEN	PRINT	REAL	STOP
THEN	INTRINSIC		

FORTRAN CONSTANTS:



1. Numeric constants

i. Integer constants:

a) An integer constant is a whole number without any decimal or fractional part.
b) This constant is a string of digits 0 to 9 & may be positive or negative.
c) No comma or any characters can be used within the digits.
d) For positive numbers use of ' + ' is optional but for negative numbers use of '-' just before the number is mandatory.
e) The magnitude of an integer constant depends upon the word size of the computer. If a computer has an n-bit word size, then the range of integer constant is -2^{n-1} to $2^{n-1} - 1$.

Examples:

$3; 14 \rightarrow$ Invalid (semi-colon is not allowed)

$+2250 \rightarrow$ Valid

$-1234 \rightarrow$ Valid

$\$55 \rightarrow$ Invalid (Dollar sign is not allowed)

ii. Real constants:

a) It is a signed and unsigned number with a decimal point. The general form of this representation is $\pm n.m$; n is the integral part and m is the fractional part.
b) The total number of digits in the fractional part depends upon the word length of the computer.
c) No comma or any characters can be used within the digits.
d) For positive numbers use of ' + ' is optional but a negative number must be preceded by a' − '.
e) Real constants can also be written in the exponential form

$$\pm n.m\ E \pm l$$

{mantissa} {exponent}

Real constants in decimal form — Signed or unsigned integers

Examples:

$1234. \rightarrow$ Valid

$41.06E + 06 \rightarrow$ Valid

$67.89E - 4.5 \rightarrow$ Invalid (Exponent cannot be a fraction)

$12,3 \rightarrow$ Invalid (Comma is not allowed)

iii. Double Precision constant:

The real constants discussed earlier are single precision real constants and have accuracy up to eight significant digits. In order to increase accuracy of a number, double precision data type is used in which number of significant digits is doubled.

E.g., The value of pie correct to 14D (i.e. 3.14285714285714) cannot be stored in a single precision constant. We need to store this value in a double precision constant in order to retain the accuracy.

Double precision constants can also be expressed as

$$\pm n.m\ E \pm l$$

{mantissa} {exponent}

double precision constants in decimal form

signed or unsigned integers

iv. **Complex constants:**

FORTRAN allows complex constants. Any complex number A+iB in FORTRAN is represented as (A, B) where A and B are real or integer constants.

2. **Character constants**:

It is a sequence or strings of characters enclosed within single quotes. The number of characters in a character constant is called the length of the character constant.

`'FORTRAN'` → Valid→ 7 character length

`FORTRAN` → Invalid (single quote is missing)

`'A+B'` → Valid→ 3 character length

3. **Logical constants**:

Two logical constants are used in FORTRAN. They are 'true' and 'false' which are written as .TRUE. and .FALSE.

VARIABLES:

i) A variable name can contain letters A to Z, digits 0 to 9 but no special characters.

ii) A variable name must start with any letters from A to Z.

iii) A variable name must not exceed a length of six characters in FORTRAN 77 but in FORTRAN 90 variable names can contain at most 31 characters and there is no reserved word which cannot be used as a variable name.

iv) Any FORTRAN key word cannot be used as a variable name.

v) Blank space in a variable name is ignored and not counted in the total number of characters.

1. **Integer variable**:

   In FORTRAN 77 any variables can be declared in two ways:

   i) by the method of type declaration (discussed later).

   ii) any variable name starts with one of the letters I, J, K, L, M, N is considered as integer variable.

   E.g., `LENGTH, MASS, JOUNIER` etc.

2. **Real variable:**

   In FORTRAN 77 any variables can be declared in two ways:

   i) by the method of type declaration (discussed later).

   ii) any variable name starts with one of the letters expecting I, J, K, L, M, N is considered as a real variable.

   e.g., `COUNT, TOTAL` etc.

3. **Double precision variable**:

   It is declared by method of type declaration.

4. **Complex variable:**
   It is declared by method of type declaration.

5. **Character constant:**
   It is defined as follow:
   `CHARACTER * N LIST`
   `LIST` indicates list of variables, separated by comma, `N` being the length of the descriptor.
   E.g.,

   ```
   CHARACTER* 10 NAME, STUDENT
   ```

   `NAME` and `STUDENT` are character variables whose values have length 10.

   ```
   CHARACTER A,B
   ```

   `A, B` are character variables whose values are of length 1.

   ```
   CHARACTER * 5 NAME,ROLL,TOTAL*3
   ```

NAME and ROLL are character variables whose values have length 5. TOTAL is a character variable having value of length 3.

```
CHARACTER*4B,C,A*3
A = 'END'
B = A
C = 'FINAL'
WRITE(*,*) A,B,C
STOP
END

Output:
'END'
'END '
'FINA'
```

```
CHARACTER SENTENCE*80,WORD(20)*16,LINE*80
SENTENCE = 'THIS IS IMPORTANT TO LEARN'
WORD(1)=SENTENCE(1:4)
I=6
J=7
WORD(2)=SENTENCE(I:J)
WORD(3)=SENTENCE((J+2):(J+10))
WORD(4)=SENTENCE(J+12:J+13)
LINE = WORD(1)//WORD(2)//WORD(3)//WORD(4)
WRITE(*,*) LINE
STOP
END

Output:
THIS            IS              IMPORTANT       TO
```

6. **Logical variables:**
   It is declared by method of type declaration.

TYPE STATEMENT:

In FORTRAN 77 any variable name started with one of the letters I, J, K, L, M, N is treated as integer variable. If we want to assign real value to these variable names, then we have to use method of type declaration. Again any variable name begins with any letters expecting I, J, K, L, M, N is considered as real variable. To make these variables integer we have to use method of type declaration.

The variables along with their types, normally declared at the beginning of the program. This is known as type declaration.

E.g.,

```
INTEGER TOTAL,SUM
```

```
REAL MASS,LENGTH

CHARACTER *10 NAME, ROLL

COMPLEX X,Y,Z

LOGICAL COUNT
```

```
LOGICAL L1,L2,L3
X=1.1
Y=2.0
Z=3.0
L1=X.GT.Y
L2= Y.LT.Z
L3=X.NE.Y
WRITE(*,*)L1,L2,L3
STOP
END

Output:
.FALSE.
.TRUE.
.TRUE.
```

ARITHMATIC OPERATORS:

Symbol	Arithmetic Operation
+	Addition
-	Subtraction
*	Multiplication
/	Division
**	Exponentiation

ARITHMETIC EXPRESSIONS:

Arithmatic Expression	Fortran Expression
a+b-c	A+B-C
ab+c	A*B+C
$a^2 + ab + b^2$	A**2+A*B+B**2
$\frac{(a+b)}{c}$	(A+B)/C
$a + b/c$	A+B/C
a^b	A**B

HIERARCHY OF OPERATIONS:

Operation	Priority
Function Evaluation	1st
Exponentiation(**)	2nd
Multiplication(*)& division(/)	3rd
Addition(+) & Subtraction(-)	4rh

LIBRARY FUNCTIONS:

Function	Operation	Type of argument	Type of result
`SIN(X)` X in radian	$\sin x$	Real	Real
`COS(X)` X in radian	$\cos x$	Real	Real
`TAN(X)` X in radian	$\tan x$	Real	Real
`ASIN(X)`	$\sin^{-1} x$	Real	Real
`ACOS(X)`	$\cos^{-1} x$	Real	Real
`ATAN(X)`	$\tan^{-1} x$	Real	Real
`SQRT(X)`	$\sqrt{x}$	Real	Real
`ALOG(X)`	$\log_e x$	Real	Real
`ALOG10(X)`	$\log_{10} x$	Real	Real
`EXP(X)`	e^x	Real	Real
`ABS(X)`	$\|x\|$	Real	Real
`AMOD(X,Y)`	Remainder of X/Y	Real	Real
`MOD(I,J)`	Remainder of I/J	Integer	Integer
`MAX0(J,K,…)`	Gives the largest of J, K,…	Integer	Integer
`MIN0(J,K,…)`	Gives smallest of J, K, …	Integer	Integer
`AMAX1(A,B,…)`	Gives the largest of A, B,…	Real	Real
`AMIN1(A,B,…)`	Gives smallest of A, B, …	Real	Real
`IFIX(X)`	Converts argument to a integer value	Real	Integer
`FLOAT(I)`	Converts argument to a real value	Integer	Real
`CMPLX(X,Y)`	Converts argument to a complex value	Real	Complex
`CABS(C)`	Gives modulus	Complex	Real
`REAL(C)`	Gives the real part of an argument	Complex	Real

AIMAG(C)	Gives the imaginary part of an argument	Complex	Real
CONJG(C)	Complex conjugate	Complex	Complex

LOGICAL EXPRESSIONS:

Logical	Meaning	Value
.NOT.	It complements the value of a logical variable.	True or False
.OR.	When operates on two variables A & B becomes true if one of A or B or both of them is true and false if both are false.	True or False
.AND.	When operates on two variables A & B becomes true only if both of A and B are true and false otherwise.	True or False

```
LOGICAL F,S,I,X

F=.FALSE.

S=.TRUE.

I=.TRUE.

X=.FALSE.
```

Logical Expression	Value
X.OR.F	.FALSE.
.NOT(S.AND.I)	.FALSE.
F.AND.I	.FALSE.
S.AND.I	.TRUE.

RELATIONAL EXPRESSIONS:

Relational Operator	Meaning
.GT.	Greater than
.GE.	Greater than or equal to
.LT.	Less than
.LE.	Less than or equal to
.EQ.	Equal to
.NE.	Not equal to

ARITHMETIC ASSIGNMENT STATEMENT:

Assignment statement is used to assign values to variables.

Syntax: Variable= Expression

This statement allows the programmer to assign the value of the expression in the R.H.S to the variable (any valid FORTRAN variable) in the L.H.S.

Statement	Comment	Remarks
`A=B+C`	Valid	Assign `B+C` to `A` (Real assignment)
`I=(J+K)/L`	Valid	Assign `(J+K)/L` to `I` (Integer assignment)
`B=SQRT(C)`	Valid	Assign `SQRT(C)` to `B` (real assignment)
`P+Q=R`	Invalid	Expression=Variable (not allowed)
`L+M=I-J`	Invalid	Expression = Expression (not allowed)
`X=Y*COS(A)`	Valid	Assign `Y*COS(A)` to `X` (Real assignment)

INPUT-OUTPUT STATEMENT:

Read Statement:

Syntax: `READ(D,F)LIST`

`D` is the input device number and is optional. Very often * is used in place of `D`. In that case the compiler understands that the data will be input through the keyboard.

`F` is the format statement label number and is optional. Very often * is used in place of `F` to indicate that the input data will be in free format.

`LIST` is the list of variable names in order in which data are to be input. The variable names are separated by commas.

Write Statement:

Syntax: `WRITE(D,F)LIST`

`D` is the input device number and is optional. `F` and `LIST` have the same meaning and same role as in case of `READ` statement.

E.g.,

```
READ(*,*) A,B
C1=A+B
C2=A-B
C3=A*B
WRITE(*,*) C1,C2,C3
Input:
A=2.0
B=3.0
Output:
C1=5.0
C2=-1.0
C3=6.0
```

FORMAT STATEMENT

Syntax: `S FORMAT(SPECIFICATION LIST)`

`S` is the format statement number which is already used in input or output statement.

E.g., `100 FORMAT('ANS=',F10.4,4X,I5)`

Different FORMAT specifications:

Specification	General form	Meaning
I specification	Iw	I indicates that the data is of integer type and w ia an integer constant indicating the field size.
F Specification	Fw.d	F represents that the data is of real type, w is an integer constant indicating the field size of the real value including sign and d indicates the number of digits appearing after the decimal point.
E Specification	Ew.d	E indicates that the data is in exponential form of real number. w is the total field width including the mantissa part, the letter E, the decimal point and sign. d is the number of digits in the mantissa part assuming that the number is in normalised exponential form.
D Specification	Dw.d	Identical with E specification and the only difference is D in place of E.
A Specification	An	A indicates that the data is of character type and n represents the field size.
X Specification	nX	N is the number of spaces to be skipped.
T Specification	Tn	Tabs to column n and prints the number at the n^{th} position.

L Specification	Lw	It is used to read or write logical variables. L specifies that the data is of logical type and w represents the field size.
Slash(/) Specification	/ or //	It is used to skip to the next line. '/' is used to print the value in the 2nd line. '//' skips the 2nd line and the value is printed in the 3rd line.

Stored value	Format Specification	Output
576729	I6	576729
576729	I10	****576729
-37216	I6	-37216
-37216	I10	****-37216
-3721.571	F9.3	-3721.571
-3721.571	F10.3	*-3721.571
157.23	F9.3	**157.230
321.38756	E15.8	*0.32138756E+03
1234567.89	E9.2	*0.12E+07
0.00001234	E10.3	*0.123E-04
NAME	A8	****NAME
1.52376214391	D17.10	*0. 1523762144D+01
PQRS	(T5,A4)	12345678910 PQRS
X=(.FALSE.)	L2	12345678910 F
9.65	(2X,F4.2)	12345678910 9.65

STOP STATEMENT:

Syntax: `STOP`

`STOP` statement is used to terminate the execution of the program.

END STATEMENT:

Syntax: `END`

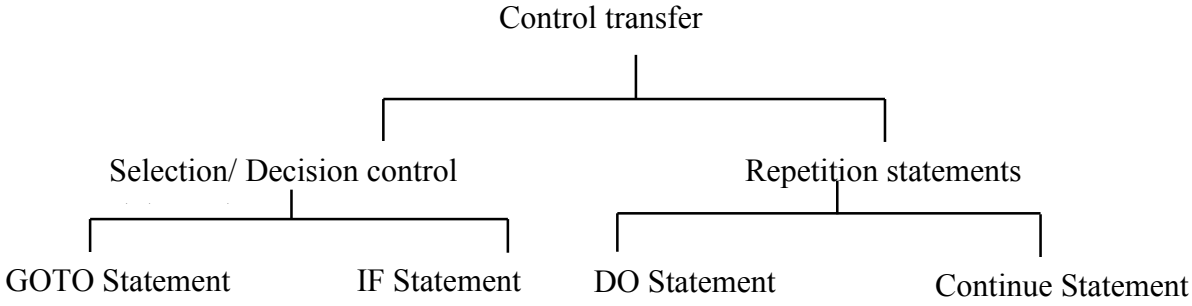
`END` statement is the last statement in a FORTRAN program and it identifies the physical end of a FORTRAN program for the compiler.

Difference:

STOP	**END**
The `STOP` statement instructs the compiler to stop the execution of the program.	The `END` statement indicates that there is no more statement left in the program unit.
The `STOP` statement can be used in anywhere in the program so that it make sense.	The `END` statement identifies the physical end of a FORTRAN program for the compiler.

The STOP statement is optional for FORTRAN 77.	The END statement is compulsory for every FORTRAN program.

CONTROL TRANSFER STATEMENT:



GOTO STATEMENT:

Syntax: `GO TO N`

`GOTO` Statement is used to transfer control the to the statement label `N`.

E.g.,

```
    I=1
10  WRITE(*,*)I
    I=I+1
    GO TO 10
    STOP
    END
```

ADDITIONAL GOTO STATEMENTS:

1. **Computed GOTO Statement:**

   Syntax: `GOTO(S1,S2,S3,…,SN),INTEGER EXPRESSION`

   `S1,S2,S3,…,SN` are statements number not necessarily distinct. If the value of the integer expression be m, then the control is transferred to the M^{th} statement i.e. `SM` is executed (M<N). If the value of the integer expression is not any one of `1,2,3,…,N,` then control is transferred to the very next statement appearing after the COMPUTED GOTO statement.

   e.g.,

   `GO TO(5,10,15,20,25),J-K`

   In this statement when `J-K=1,2,3,4,5,` program control is shifted to the statement `5,10,15,20,25` respectively. When `J-K` is less than 1 or more than 5 then control is transferred to the very next statement appearing after the `COMPUTED GO TO` statement.

2. **Assigned GO TO statement:**

Syntax: `ASSIGN` Integer constant `TO` Integer variable

`GOTO` Integer variable, (`S1,S2,S3,…,SN`)

The ASSIGN statement initializes an integer value to the integer variable. The integer variable is used in assigned `GO TO` statement and if the value of the variable matches with any one of the list of the statements numbers `S1,S2,S3,…,SN` of the assigned `GO TO` statement then the control is transferred to that statement for execution otherwise an execution error occurs.

E.g.,

```
ASSIGN 44 TO I
GOTO I,(10,44,20,25)
```

At first it assign 44 to I and then in the assigned GOTO statement the control is transferred to the statement number 44.

IF STATEMENTS:

1. Logical IF Statement:

Syntax: `IF(`Logical/Relational expression`)` Statement

Logical IF statement first evaluate the Logical/Relational expression and then execute the statement associated with it if the value of the expression is true. If the logical/Relational expression is false, the Logical IF statement is not executed and control is transferred to the very next statement after it.

E.g.,

```
WRITE(*,*)'GIVE X'
READ(*,*) X
Y=1.0
IF (X.LE.5) Y=0
Z=X+Y
WRITE(*,*) Z
STOP
END
Input:
2.0
Output:
2.0
```

2. Arithmetic IF Statement:

Syntax: `IF(`Arithmetic Expression`) S1,S2,S3`

Arithmetic expression (any valid FORTRAN arithmetic expression) is evaluated first. The control is transferred to `S1,S2,S3` according to the value of arithmetic expression is negative, zero or positive.

E.g.,

```
WRITE(*,*)'GIVE X,Y'
READ(*,*) X,Y
IF (X-Y) 10,20,30
```

```
10 Z=Y-X
   GO TO 40
20 Z=0.0
   GO TO 40
30 Z=X-Y
   GO TO 40
40 WRITE(*,*) Z
   STOP
   END
   Input:
   2.0 3.0
   Output:
   1.0
```

3. **Block IF Statement:**

a) **IF-THEN-ENDIF Statement:**

Syntax:

```
IF(CONDITION)THEN
    S1
    S2
    .
    .
    .
    SN
ENDIF
```

`S1,S2,S3,…,SN` is a set of executable statement. If the condition is true, then the statements are executed and when all the statements are executed, control is transferred to the very next statement after `ENDIF`.

E.g.,

```
READ(*,*)A,B
IF(A.GT.B)THEN
T=B
B=A
A=T
ENDIF
WRITE(*,*)A,B
STOP
END
```

---

b) **IF-THEN-ELSE-ENDIF Statement:**

Syntax:

```
IF(CONDITION)THEN
    S1
        S2
        .
        .
        .
        SN
```

```
ELSE
        N1
        N2
        .
        .
        .
        NM
    ENDIF
```

`S1,S2,S3,…,SN` is a set of executable statement. `N1,N2,N3,…,NM` is a set of another executable statement. After the execution of the condition (relational/logical expression), the set of statements `S1,S2,S3,…,SN` are executed only if the condition is true. The set of statements `N1,N2,N3,…,NM` are executed i.e. else block is countered otherwise.

E.g.,

```
READ(*,*)A,B
IF(A.GT.B)THEN
LARGEST=A
ELSE
IF(B.GE.A)THEN
LARGEST=B
ENDIF
WRITE(*,*)LARGEST
STOP
END
```

IF-ELSE-IF STATEMENT:

Syntax:

```
IF(CONDITION 1)THEN
     BLOCK 1
ELSEIF(CONDITION 2)THEN
     BLOCK 2
          .
          .
          .
 ELSEIF(CONDITION M)THEN
      BLOCK M
ELSE
      BLOCK M+1
```

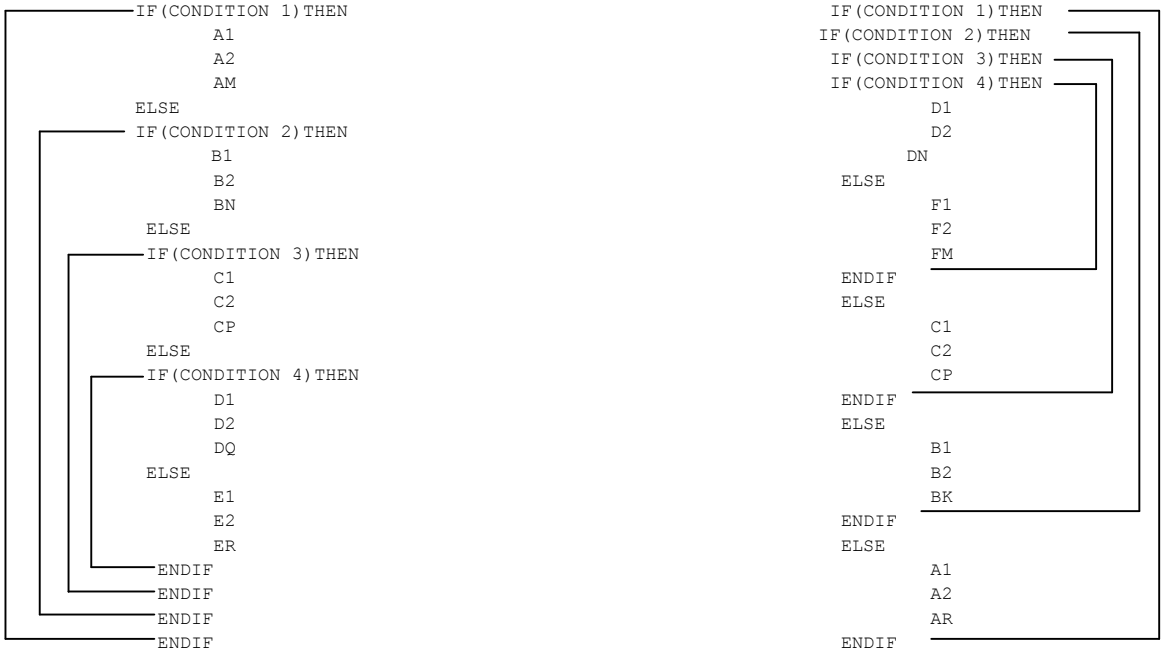
```
ENDIF
```

This IF structure allow the programmer to select one from more than two alternatives. At first the condition 1 is evaluated, if it is true then block 1 is executed and then the control is transferred to the next statement after ENDIF. If condition 2 is true then block 2 is executed and the control is transferred to the next statement after ENDIF. Similarly for other blocks.

E.g.,

```
READ(*,*)A,B,C
IF((A.GE.B).AND.(A.GE.C))THEN
LARGEST=A
ELSEIF((B.GE.A).AND.(B.GE.C))THEN
LARGEST=B
ELSEIF((C.GE.A).AND.(C.GE.B))THEN
LARGEST=C
WRITE(*,*)LARGEST
STOP
END
```

NESTED BLOCK IF STRUCTURE



DO STATEMENT:

Syntax: `DO K CONTROL VARIABLE=START VALUE,STOP VALUE,STEP`

```
                    S1
                     S2
                     .
                     .
                     .
                     SN
      K CONTINUE
```

K is the statement label of a FORTRAN statement. Generally against statement number K the continue statement is written. `CONTROL VARIABLE` may be integer, real number or double precision. `START VALUE` is the initial value of the control variable. `STOP VALUE` is the terminal value of the control variable and `STEP` is the increment of the control variable after each iteration. The value of the increment may be positive or negative but must not be zero. `START VALUE` and `STOP VALUE` and this value may be constant (real or integer) or variable or expression.

The above form of DO statement though most general, may not be supported by some of the older versions of the compiler. The following form is however supported by all compilers

```
DO N I=START VALUE,STOP VALUE,STEP

                    S1
                     S2
                     .
                     .
                     .
                     SN
      N CONTINUE
```

where `START VALUE,STOP VALUE,STEP` are integers.

```
      DO 10 K=2,100,2
      WRITE(*,*)K
   10 CONTINUE
      STOP
      END
```

SUBSCRIPTED VARIABLES:

1. **One-dimensional arrays:**
   An one-dimensional array refers to a single variable which has several values known as elements. Each array has a name and each element of that array is identified by a positive integer written within the parenthesis after the array name. For example

```
A(20)
```

declares an one dimensional array of length 20 consisting of 20 real numbers stored contiguously in memory. By convention FORTRAN arrays are indexed from 1 and up. Thus the first number in the array is denoted by `A(1)` and the last by `A(20)`.
However, arbitrary index ranged arrays can be in the following way:

```
B(0:19),C(-162:237)
```

Here B is an one dimensional array of length 20, the index runs from 0 to 19. C is also an one dimensional real array of length 237-(-162) +1 = 400, the first element is denoted by `C(-162)` and last element is `C(237)`.

2. **Multi-dimensional arrays:**

An array where the elements can be referred by two or more than two subscripts is known as multi-dimensional array.
Matrices are usually represented by two-dimensional arrays. For example, the declaration

```
A(3,5)
```

defines a two-dimensional array of 3*5=15 real numbers. It is useful to think of the first index as the row index, and the second as the column index. Hence we get the graphical picture:

```
(1,1)  (1,2)  (1,3)  (1,4)  (1,5)
(2,1)  (2,2)  (2,3)  (2,4)  (2,5)
(3,1)  (3,2)  (3,3)  (3,4)  (3,5)
```

Two-dimensional arrays can be defined for arbitrary ranges. The general syntax for this declaration is

```
NAME (LOW_INDEX1 : HI_INDEX1, LOW_INDEX2 : HI_INDEX2)
```

The total size of the array is then

SIZE = (HI_INDEX1-LOW_INDEX1+1)*(HI_INDEX2-LOW_INDEX2+1)

Similarly an array `A(4,5,6)` is known as three-dimensional array.

THE DIMENSION STATEMENT:

Before using any subscripted variable in a program the compiler must be provided i) name of the subscripted variable, ii) number of subscripts, i.e. whether it is one-dimensional or it is two-dimensional.

Syntax: `DIMENSION ARRAY1(SIZE),ARRAY2(SIZE),...`

Array size is an integer constant not a variable. However, DIMENSION statement is not required if the array is declared by the type declaration statement. For example

```
REAL A(10),B(7,6)
```

```
INTEGER I(0:5,0:4)
```

E.g.,

```
      REAL A(3,5)
      INTEGER I,J
      DO 20 J = 1, 3
      DO 10 I = 1, 3
      A(I,J) = REAL(I)/REAL(J)
   10 CONTINUE
   20 CONTINUE
      DO 30 I = 1,3
      WRITE(*,*)(A(I,J),J=1,5)
   30 CONTINUE
      STOP
      END
```

DATA STATEMENT:

Syntax: `DATA V1,V2,V3,…,VN/D1,D2,D3,…,DN`

here `V1,V2,V3,…,VN` are list of variables whose values are `D1,D2,D3,…,DN` respectively.

`DATA A,B,C /1,2,3`
declares `A=1,B=2,C=1.`

PARAMETER STATEMENT:

Syntax: `PARAMETER(NAME 1=U1,NAME 2=U2,…,NAME K=UK)`

The parameter statement assigns a value to a constant which cannot be changed by any statement in the program. This parameter statement is placed before all executable statements.

SUBPROGRAM:

There are mainly two types of subprograms in FORTRAN.

1. **Function subprogram:**
   It is program unit separated from the main program and can be called by the main program. Function subprogram is used when only one value is required to be returned from the sub program .

   Syntax: Type `FUNCTION` name(argument list)

```
----------------------------
----------------------------
name=expression
----------------------------
----------------------------
RETURN
----------------------------
END
```

Type indicates the type of the function (real, integer, logical). If type indicator is omitted then function type is defined by the first letter of the function name. Function name is any valid FORTRAN variable name. In the argument list all the arguments are to be separated by commas. There can be more than one RETURN statement in a subprogram. The program control is transferred to the main program by the use of RETURN statement.

2. **Subroutine subprogram:**

Syntax: SUBROUTINE name (argument list)

When more than one value is to be returned from the subprogram to the calling program, subroutine subprograms are used. SOUBROUTINE name is any valid FORTRAN variable name. The name of SOUBROUTINE returns no value. That is why type declaration convention is not allowed while naming a SUBROUTINE. All arguments in the argument list are separated by comma. The values to be returned to the calling program, take place through the arguments.

e.g.,

```
     READ(*,*)A,B,C
     CALL AREA(A,B,C,DELTA)
     WRITE(*,100)A,B,C,DELTA
100  FORMAT(F10.6,2X,F10.6,2X,F10.6,2X,F10.6)
     SUBROUTINE AREA(A,B,C,DELTA)
     S=A+B+C
     DELTA=SQRT(S*(S-A)*(S-B)*(S-C))
     RETURN
     END
```

OPENING OF A FILE:

The instruction for opening a file has the following form:

```
OPEN(UNIT NO.,FILE='FILENAME',STATUS='NEW OR OLD')
```

FILE NAME is the name of the file. UNIT NO. is non-negative integer that is unique for each file used in program. STATUS indicates the nature of the file i.e. whether it is existing one or it has to be created. For input files the status will be OLD and for output files status will be NEW.

e.g.,

```
OPEN(5,FILE='X.IN',STATUS='OLD')
```

CLOSING A FILE:

The instruction for closing a file has the following form:

```
CLOSE(UNIT NO.)
```

Normally this is done at the end of the program before STOP statement.

E.g.,

```
CLOSE(5)
```

RULES FOR WRITING A PROGRAM IN FORTRAN

1. One statement should be typed in one line.
2. The statements should be written from 7th column and it should extend up to 72th column.
3. Normally there are 80 columns in a line. 73rd to 80th columns are ignored by the compilers.
4. First five columns are kept reserve for writing statement numbers. The ranges of statement numbers me be from 1 to 99999.
5. Sixth column is used for writing the continuation numbers. Sometimes long instructions may not be accommodated within 72th column. In such a situation the instruction extends to subsequent lines through the use of a continuation number. For first continuation line the number is 1, for 2nd it will be 2 and so on.