Chapter-14

Introduction to FORTRAN

14.1 Introduction

The FORTRAN is an acronym for FORmula TRANslation. It is one of the oldest computer languages, created by John Backus and released in 1957. It was initially designed for scientific and engineering computations. It is a most widely used high level programming language and is being used mainly for mathematical or numerical or scientific applications. It has been revised several times from first introduced date to now. By convention, a FORTRAN version is denoted by the last two digits of the year when the standard was proposed. The history of FORTRAN's versions can be classified as follow:

- FORTRAN 1957
- FORTRAN II
- FORTRAN IV
- FORTRAN 66 (released as ANSI standard in 1966)
- FORTRAN 77 (ANSI standard in 1977)
- FORTRAN 90 (ANSI standard in 1990)
- FORTRAN 95 (latest ANSI standard version)

The most common and popular version of FORTRAN today is still Fortran 77, although FORTRAN 90 is growing in popularity. FORTRAN 95 is a revised version of FORTRAN 90.

14.1.1 Characteristics of FORTRAN

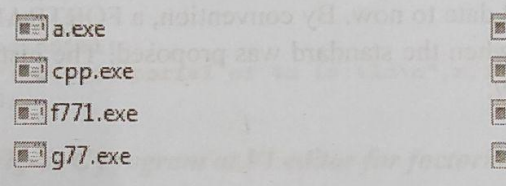
i. Its numerical capacity and input/output facilities are more powerful than others. It is dominating language for mathematical computing.
ii. It is simple to understand and thus easy to learn.
iii. Programs written in FORTRAN are more portable than those in other programming languages.
iv. The program written in FORTRAN is Machine Independent (i.e. it is easy to translate a program from one machine to another).
v. It is standardized by ANSI and ISO.
vi. It has rich library for built in functionalities.

14.1.2 Limitations of FORTRAN 77

i. The `struct` and `class` is not available in F77. The F77 has not concept of virtual function and template although F90/95 has concept of these facilities.
ii. No Dynamic Memory Allocation
iii. Code is difficult to extend, maintain and reuse
iv. Poor string handling capabilities
v. Loop controls are limited. The use of goto statement is high.

14.2 Steps in writing program in source Code

We need editor and compiler to test a sample program. For learning purpose, we will use Force 2.0 FORTRAN 77 compiler. [You can download Force 2.0 compiler from link: `http://www.winsite.com/fortran/fortran+77+uestudio/freeware/` and install in your computer]. After installation of Force 2.0 Compiler, you see following .exe files (i.e. commands) in C:\Program Files\Force 2.0\Bin directory (as default directory).



Here, `g77.exe` is compiler which is used for compilation of FORTRAN program and `a.exe` is used to execute compiled program.

Let us consider a FORTRAN program say `Hello.f` is saved within `D:\FORTRAN\` directory. Then, it is compiled and executed as following:

Step 1: Open MS-DOS [In window 7, Click on `Start->Accessories→Right Click on Command Prompt→ Run As Administrator→Yes`]

Step 2: Change directory in `C:\Program Files\Force 2.0\Bin` (i.e. default directory)

Step 3: Write a program say `Hello.f` in Force Compiler and Save into `D:\FORTRAN\` directory

Step 4: Compile the program using MS-DOS window

```
C:\Program Files\Force 2.0\Bin>g77.exe "D:\FORTRAN\Hello.f"
```

Setep 5: Execute the program using MS-DOS window

```
C:\Program Files\Force 2.0\Bin>a.exe "D:\FORTRAN\Hello.f"
```

14.2.1 Writing Source Code

Computer instructions (i.e. source code) in FORTRAN are written in a text editor. The source code in FORTRAN can be written using any text editor such as `Notepad` or any specific editor developed for FORTRAN. As FORTRAN 77 is not a free-format language, it has a very strict set of rules for writing source code. The position of instruction in column is very important. The column position rule is as following:

i) **Column 1 : Blank, or a "c" or "*" for comments.**

i.e. in first column, we can either write `c` or `*` character for comment or leave blank. Thus, we can't start writing any other instructions from the very beginning of the editor.

1	2-5	6	7-72	73-80
c			`This is program to calculate area of a cirlcle`	

ii) **Column 2-5: Statement label (optional)**

i.e. In column 2-5, we can write some text to denote or label the particular statement. For example:

```
900   write(*,*)'This is statement which is labeled as 900'
```

In above line, `900` is label name which is used to identify the particular statement. We can write characters also for label name.

1	2-5	6	7-72	73-80
	900		`write(*,*)'This is statement which is labeled as 900'`	

iii) **Column 6 : Continuation of previous line (optional)**

Sometimes, a statement does not fit into the 66 available columns (i.e. 7-72 columns) of a single line (i.e. statement having length more than 66). We have to then break the statement into two or more lines, and use the continuation mark in position 6. For example: if a single statement like `write(*,*),'This is statement which is labeled as 900'` is written in two lines of editors, then continuation mark is written in column 6.

1	2-5	6	7-72	73-80
		+	`Write(*,*),'This is statement which is labeled as 900'`	

Here, '+' sign in above code denotes continuation mark that means both lines represent single statement, not two separate statements. Any character can be used instead of the plus sign as a continuation character. It is considered good programming style to use the `plus sign` or an `ampersand`, or `digits` (using 2 for the second line, 3 for the third, and so on).

iv) **Column 7-72: Statements**

In this position, we write actual instructions or statements for execution.

v) **Column 73-80: Sequence number (optional, rarely used today)**

Example 14.1

Write a program in FORTRAN to display "Hello Word" in screen.

1	2-5	6	7-72	73-80
c			`program Hello Word`	
c			`The hello word program to display message in screen`	
			`This program is written by R.D. Bhatta`	
			`write (UNIT=*,EMT=*) 'Hello Word'`	
			`Stop`	
			`End`	

In above sample program, the PROGRAM statement gives a name to the program and declares that it is a main program unit. The second statement is known as comment. A line that begins with the letter 'c' or an asterisk in the first column is a comment. Comments may appear anywhere in the program. We can use FORTRAN programs that use the exclamation mark (!) for comments. This is not a standard part of FORTRAN 77, but is supported by several FORTRAN 77 compilers and is explicitly allowed in FORTRAN 90.

The write statement is used to display some text in screen or put to a file. The parentheses enclose a list of control items which determine where and in what form the output appears. Here, UNIT=* selects the standard output device which is computer monitor by default; FMT=* sets a default output layout i.e. default format. Thus, character asterisk (i.e. *) is used is used to select a default or standard option. The write statement can be shortened as write (*,*) 'Hello Word'

The last statement is used to inform the compiler about ending of the code. We can use stop command at the end of program. The stop statement is optional and the program will stop when it reaches the end anyway, but it is recommended to always terminate a program with the stop statement to emphasize that the execution flow stops there.

The column position rule can be illustrated with following another sample program which reads radius of a circle and display its area with field width 10 and precision 2.

Example 14.2

Write a program in FORTRAN to read radius of a circle and find its area with field width 10 and precision 2.

1	2-5	6	7-72	73-80
			program circle	
			real r, area	
c			This program reads a real number r and prints	
c			the area of a circle with radius r.	
			write (*,*) 'Enter Radius of Cirlce:'	
			read (*,*) r	
			area = 3.14159*	
		+	r*r	
			write (*,900) 'Area = ', area	
	900		format (A10,F10.2)	
			Stop	
			end	

14.2.2 Compilation and Linking

The main function of a FORTRAN compiler is convert source code into object code and generate corresponding object file. After compilation, the linker just takes the set of object modules produced by the compiler and links them all together into an executable image. One of

these modules must correspond to the main program unit; the other modules will correspond to procedures and to block data subprogram units. In Force 2.0 compiler, the FORTRAN program can be compiled with following command.

```
C:\Program Files\Force 2.0\Bin>g77.exe hello.f
```

14.2.3 Execution

After compilation and linking process, executable program is generated. The generated executable program is then loaded into memory and run to produce corresponding output. In Force 2.0 compiler, the FORTRAN program is executed with following command.

```
C:\Program Files\Force 2.0\Bin>a.exe hello.f
Output
Hello Word
```

The steps required to compile, link and run a FORTRAN program can be visualized with following diagram.

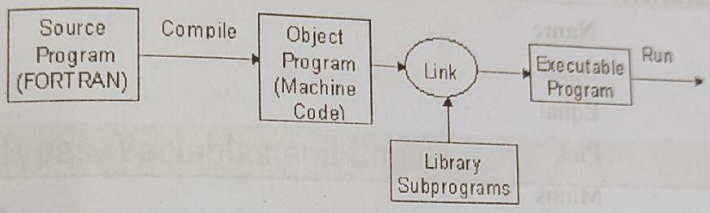


Fig 1.1: Steps in writing and execution of a FORTRAN program

14.3 Structure of a FORTRAN program

A FORTRAN program generally consists of a main program and subprograms (procedures or subroutines or functions). The subprograms will be illustrated in later part. The structure of a main program is:

a) Program Name
b) Declaration Section
c) Statements
d) Stop
e) End

The program name section begins with keyword `program` and it helps to assign a name to a program which will be useful for future reference. The second section `declaration` allows defining variables and parameters required in program. We write actual statements in third section i.e. `statement section`. The keyword `end` is used to inform the compiler about ending of the code. The `stop` statement is optional and may seem extra statement since the program will stop when it reaches the end anyway, but it is recommended to always terminate a program with the `stop` statement to emphasize that the execution flow stops there.

14.4 Character Set

The set of characters that are used to form words, numbers and expression in FORTRAN is called FORTRAN character set. The combination of these characters form words, numbers and expressions in FORTRAN program. The FORTRAN character set consists of 26 uppercase and 26 lowercase letters (i.e. alphabetic characters), the numbers 0 through 9 (i.e. digits), and special characters.

The complete character set is

i. Letters

```
Uppercase Letters: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z
Lowercase Letters: a b c d e f g h i j k l m n o p q r s t u v w x y z
```

ii. Digits:

```
0 1 2 3 4 5 6 7 8 9
```

iii. Special Characters:

Character	Name
	Blank
=	Equal
+	Plus
-	Minus
*	Asterisk
/	Slash
(	Left parenthesis
)	Right parenthesis
,	Comma
.	Decimal point
$	Currency symbol
'	Apostrophe
:	Colon
!	Exclamation point
_	Underscore
"	Double quote

iv. Escape Sequences

An escape sequence is a non-printing characters used in FORTRAN. It is a character combinations consisting of a backslash (\) followed by a letter or by a combination of digits. It is as a single character and is therefore valid as a character constant. The following table lists some escape sequences and they are compatible with the C programming language.

Escape Sequence	Meaning
\n	New Line
\t	Tab
\b	Backsapce
\f	Form Feed
\0	Null
\'	Single Quote
\"	Double Quote
\\	Slash

Example 14.3

Write a program in FORTRAN to illustrate the use of escape sequence.

```
      program escape sequence
c     This is the program to illustrate the use of escape sequence
      write (UNIT=*, FMT=*) '\tI\n\tLove\n\tYou'
      end
```

Output

```
        I
    Love
    You
```

14.5 Data Types, Variables and Constants

14.5.1 Data Types

There may exist various types of data in programming environment. To store such type of data, we need variables. It is good practice to define every variable before its use. The FORTRAN program supports following standard data types.

Data type	Description
Integer	Represents both positive and negative integral (non-fractional) numbers. It occupies 4 bytes of memory in RAM and can store integer numbers from -2^{31} to $+2^{31}$.
Real	Represents both positive and negative numbers with a fractional part. It occupies 4 bytes of memory in RAM and can store values from $1.2*10^{-38}$ to $3.4*10^{38}$.
Double Precision	same as Real but using twice the storage space and greater precision. It occupies 8 bytes of memory in RAM and can store values from $2.2*10^{-308}$ to $1.8*10^{308}$.
Complex	Represents an ordered pair of REAL data: real and imaginary components. It occupies 2*4 bytes=8 bytes of memory in RAM.
Logical	Represents the Boolean data representing TRUE or FALSE (i.e. either 1 or 0). It reserves only one bit of memory.
Character	Represents one alphanumeric character. It reserves only one byte of memory. The character*N represents multiple characters (i.e. string with N characters) and occupies N bytes of memory in RAM..

It is important to understand that the number of memory bytes occupied by a particular data type and its range is not fixed for all machines (i.e. it may differ one machine to another machine).

Example 14.4

Write a program in FORTRAN to read two integers from user and display their sum in screen.

```
program variables
integer a,b,c
write (*,*) 'Enter two integers:'
read(*,*) a,b
c=a+b
write(*,*) 'The sum of two integers:',c
end
```

```
Output
Enter two integers:10  20
The sum of two integers: 30
```

Example 14.5

Write a program in FORTRAN to read two fractional numbers from user and display their sum in screen.

```
program variables
real a,b,c
write (*,*) 'Enter two fractional numbers:'
read(*,*) a,b
c=a+b
write(*,*) 'The sum of real numbers:',c
end
```

```
Output
Enter two fractional numbers:3.5   6.8
The sum of real numbers:  10.3000002
```

Example 14.6

Write a program in FORTRAN to read two complex numbers from user and display their sum in screen.

```
program variables
complex a,b,c
write (*,*) 'Enter two complex numbers:'
read(*,*) a,b
c=a+b
write(*,*) 'The sum of complex numbers:',c
end
```

```
Output
Enter two complex
numbers:
(1,3)
(3,8)
The sum of complex
numbers: (4.,11.)
```

It is important to note down that the complex number should be given in format: small bracket open, real part, comma, imaginary part and small baracket close.

Example 14.7

Write a program in FORTRAN to read a string and display on screen

```
program variables
character(LEN=20) name
write (*,*) 'Enter your name:'
read(*,*) name
write(*,*) 'Your name is:',name
end
```

```
Output
Enter your name: Ram
Your name is: Ram
```

In above example, we can't input string with space. To read a string with space, we have modify read statement as

```
read(*,'(A)') name
```

14.5.2 Variables

A variable is an identifier with a name, data type, and value. It's type depends upon either data type provided at the time of its declaration (i.e. explicit definition) or first character in its name (i.e. in the case of implicit definition). The variable name is a symbolic name of the data item and must conform to the rules given for identifier's names. A variable cannot be used or referred to unless it has been defined through an assignment statement, input statement, data statement, or through association with a variable or array element that has been defined

Defining data types for a variable

i. **Implicit definition**

We can define a variable without explicit data type in variable declaration statement. If not explicitly specified by a type statement, the data type of a data item is determined implicitly by the first character of its name. By default, symbolic names beginning with I, J, K, L, M, or N (uppercase or lowercase) imply an integer data type; names beginning with all other letters imply a real data type.

ii. **Explicit Definition**

We can define a variable with exact type using following syntax:

```
type_name variable_name
```

e.g.
```
integer num;
real a,b,c
```

14.5.3 Constants

A constant has a value which is fixed when the program is written. It is called constant as its value doesn't change during the program execution.

Classification of Constants

i. **Integer Constants**

The general form of an integer constant is a sign (plus or minus) followed by one or more digits. If the number is positive the plus sign is optional.

Here are some examples of valid integer constants

```
-10 420 0 +10476
```

ii. **Real Constants**

A real constant contains a decimal point or an exponent (or both) to distinguish it from one of integer type. The letter "E" is used in FORTRAN to represent "times 10 to the power of". For example, the constant 1.234 ×105 is written as "1.234E-5".

Here are a few examples of valid real constants:

```
2.5 -100. 1E3 +13.456E4 .0001
```

iii. Complex Constants

A complex constant has the form of two real or integer constants separated by a comma and enclosed in a pair of parentheses. The first number is the real component and the second the imaginary component.

Some examples of valid complex constants are:

```
(2.14,-3.67)  (+1E2,0.105)  (0,0)  (-0.99,2.78E5)
```

iv. Logical Constants

There are only two possible logical constants, and they are expressed as: .TRUE. and .FALSE. The dots at each end are needed to distinguish these special forms from the words TRUE and FALSE, which could be used as symbolic names.

v. Character Constants

A character constant consists of a string of characters enclosed in a pair of apostrophes which act as quotation marks. Within the quoted string, any characters available in the character set of the programming language are permitted.

Examples of valid character constants are:

```
'A'        'Ram'        'This is a constant including spaces'
```

14.6 Operators and Expression

An `operator` is a symbol that operates on a certain data type or data item. Operators are used in programs to perform certain mathematical or logical manipulations. For example, in a simple expression `8+9`, the symbol `+` is called an `operator` which operates on two data items `8` and `9`. The data items that operators act upon are called `operands`. Here, `8` and `9` are `operands`.

An `expression` is a combination of variables, constants and operators written according to syntax of the language. For example, the following are examples of expressions –

```
8+9   a+b*c    a>b    a*b/3
```

Types of operators

a) Arithmetic Operators
b) Relational Operators
c) Logical Operators
d) Assignment Operators
e) Concatenation Operators

14.6.1 Arithmetic Operators

An arithmetic operator is used to evaluate a numeric expression. Evaluation of an arithmetic expression produces a numeric value. The arithmetic expressions may be formed by using one or more arithmetic operands together with arithmetic operators and parentheses. Arithmetic operands must identify values of type integer, real, double precision, or complex.

Associativity	Precedence	Operation	Symbol	FORTRAN Expression	Arithmetic Expression
right to left	Highest	Exponentiation	**	A**B	ab
left to right	High	Multiplication and Division	*	A*B	a×b
			/	A/B	a÷b
left to right	Low	Addition, Subtraction, and Unary Minus	+	A+B	a+b
			-	A-B	a-b
				-A	-a

Operator Precedence and Associativity

Each operator in FORTRAN has a precedence related with it. This precedence is used to determine how an expression involving more than one operator is evaluated. There are distinct levels of precedence and an operator may belong to one of these levels. The operators at the higher level of precedence are evaluated first. The operators of the same precedence are evaluated either from 'left to right' or from 'right to left', depending on the level. This is known as the associativity property of an operator.

Example 14.8

What is the output of the following program?

```
  program precedence
  real x,y,z,result
  x=10
  y=6.5
  z=15.4
  result=y+z*x**2
  write(*,900) 'The value of result:',result
900 format(A20,F10.2)
  stop
  end
```

Output

```
The value of result:   1546.50
```

Explanation:

```
Step 1: x**2=10**2=102=100
Step 2:
z*x**2=y+z*100=15.4*100=1540
Step 3:
y+z*x**2=y+1540=6.5+1540=1546.50
```

Division Rule

The result of division operator (i.e. /) depends upon the data type of operands. If the operands are both integers, an integer division is performed and the result is always integer value, otherwise a real arithmetic division is performed.

Type First Operand	Type of Second Operand	Type of quotient
Integer	Integer	integer
Real	Real	Real
Integer	Real	Real
Real	Integer	Real

Example 14.9

What is the output of the following program?

```
      program division
      integer a,b
      real c
      a=10
      b=4
      c=a/b
      write(*,900) 'The value of c:',c
900   format(A20,F10.2)
      stop
      end
```

```
Output
The value of c:          2.00
```

Explanation:
When an integer 10 is divided by another integer 4, the result is also an integer. So, it produces 2 as quotient not 2.5. The integer result is assigned to real variable and displayed with precision 2. It's output will be 2.00.

14.6.2 Relational Operator

Relational operators are used to compare two similar operands, and depending on their relation, take some actions. Relational operators compare their left hand side operand with their right hand side operand for lesser than, greater than, lesser than or equal to, greater than or equal to, equal to or not equal to relations. The value of a relational expression is either 1 (if condition is true) or 0 (if condition is false). FORTRAN supports following relational operators.

FORTRAN Operator	Common Operator	Meaning	Example	Output (int a=20, b=6)
.LT.	<	Less than	a.LT.b	0
.GT.	>	Greater than	a.GT.b	1
.LE.	<=	Less than or equal to	a.LE.b	0
.GE.	>=	Greater than or equal to	a.GE.b	1
.EQ.	==	Equal to	a.EQ.b	0
.NE.	!=	Not equal to	a.NE.b	1

14.6.3 Logical Operators

Logical operators are used to compare or evaluate logical and relational expressions. The operands of these operators must produce either `1 (true)` or `0 (false)`. The whole result produced by these operators is also either true or false. The FORTRAN supports `AND`, `OR` and `NOT` operators and dot is written before and after them.

A	B	.NOT. A	A .AND. B	A .OR. B
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.
.TRUE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.
.FALSE.	.TRUE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.

14.6.4 String Concatenation Operator

Two CHARACTER strings can be joined together in a single called `concatenation operator`. The concatenation operator is represented by a double forward slash `//`. It is also known as `character operator`.

Example 14.16

Write a program to read first name and family name from user and concatenate two parts of name to display full name.

```
program string concatenation
character fname*10,lname*10,name*20
write(*,*) 'What is your first name?'
read(*,*) fname
write(*,*) 'What is your family name?'
read(*,*) lname
name=fname//' '//lname
write(*,*) 'Your full name is:',name
stop
end
```

```
Output
What is your first name? Ram
What is your family name? Thapa
Your full name is:Ram Thapa
```

14.6.5 Assignment Operators

Assignment operators are used to assign the result of an expression to a variable. The usual assignment operator is '='.

For example: Let us consider following expression

```
X=X+10
```

In above example, 10 is added to previous value of X and the result is assigned to X. with the help of assignment operator i.e. '='. The precedence of assignment operator is least and the assiciativity is right to left.

14.7 Library Functions

There are two types of functions in FORTRAN like other programming languages: Library Functions and User-defined functions. The user-defined functions can be defined by user while library functions are built-in functions. The library functions can be directly called in program to make program development easy. Some of the library functions available in FORTRAN can be listed as below.

Abs()	:	To calculate absolute value
Min()	:	To calculate minimum value among multiple numbers
Max()	:	To find maximum value among multiple numbers
Sqrt()	:	To calculate square root of a number
Sin()	:	To calculate sine value
Cos()	:	To calculate cosine value
Tan()	:	To calculate tangent value
Atan()	:	To calculate inverse of tangent
Exp()	:	To calculate exponential (natural) of a number
Log()	:	To calculate logarithm (natural) of a number
Read()	:	For reading data from input device
Write()	:	For writing data to output device

Example 14.11

Write a program to read three numbers and find minimum number using library function.

```
program library function
integer a,b,c
write(*,*) ,'Enter three numbers:'
read(*,*),a,b,c
write(*,*),'The minimum number is:',Min(a,b,c)
stop
end
```

```
Output
Enter three numbers: 10  45  5
The minimum number is: 5
```

Example 14.12

Write a program to read a number form user and find square root of the number

```
      program square root
      real num
      write(*,*) ,'Enter a number:'
      read(*,*),num
      write(*,100),'The square root is:',Sqrt(num)
100   format(A20,F5.2)
      stop
      end
```

```
Output
Enter a number:25.0
The square root is: 5.00
```

14.8 Formatted and Unformatted Input/Output Statements

14.8.1 I/O Functions

I/O functions are required in any computer program to handle input and output activities. The FORTRAN has two functions for I/O: `read()` and `write()`. The `read()` function is used for transferring data from input device to main memory of the computer. Similarly, the `write()` function is used to transfer data from main memory of computer (i.e. program) to output device.

Syntax:

```
read (UNIT=unit_no, FMT=format_no) list-of-variables-separated-by-comma
write(UNIT=unit_no, FMT=format_no) list-of-variables-separated-by-comma
```

Here, `unit_no` is an integer that represents input or output device. A computer may have multiple input and output devices such as keyboard, magnetic tape, card reader, monitor, printer etc. In this case, a unique number is given to each device. An `asterisk (*)` in `read()` function represents `stdin` i.e. the keyboard and in `write()` function it represents `stdout` i.e. monitor. So, we can either specify a particular device number in `UNIT` field or * for default device.

The `format_no` refers to a label for a format statement. We can write an `asterisk (*)` to indicate default formatting (i.e. `list-directed I/O`.)

The list-directed input uses following rules:

i. The input values are separated by strings of blanks or a comma.

ii. The input values, except for character strings, cannot contain blanks.
iii. The character strings can be quoted strings, using pairs of quotes ("), pairs of apostrophes (), or unquoted strings.
iv. The complex numbers are given as two real numbers separated by a comma and enclosed in parentheses.
v. The input data items can be preceded by repetition counts, as in:
4*3 (to input 3 four times)

Similarly, the list-directed output uses following rules

i. The character strings are printed as they are. They are not enclosed in quotes.
ii. The real number is displayed with rounded off.

The read and write statement with default unit and format can be written as following:

```
read (*,*)list-of-variables-separated-by-comma
write(*,*)list-of-variables-separated-by-comma
```

The comma at beginning of the variables-list is optional. The syntax will be as following if comma is used.

```
read (*,*),list-of-variables-separated-by-comma
write(*,*),list-of-variables-separated-by-comma
```

14.8.2 Format statements

We can specify some particular input or output format, e.g. how many decimals in real numbers. FORTRAN has some rules for formatted input and output while inputting and outputting data. When data are to be input into program or the results to be output from program, we can mention the type of data (i.e. integer, real or character etc) and its size (i.e. format specification). The specification of the types of the data and its size is called FORMAT specification.

The syntax for FORMAT statement is:

```
Label FORMAT(format_specification1, format_specification2,.....)
```

Where

Label is used to identify correct format specification while reading or writing a particular data. The different format specifications are separated by commas in single format statement.

The most common format code letters are:

i. I - integer
ii. A - text string
iii. D - double precision numbers, exponent notation
iv. E - real numbers, exponent notation
v. F - real numbers, fixed point format
vi. X - horizontal skip (space)
vii. / - vertical skip (newline)

i. I Format

The symbol `I` is used to denote the integer data. The general format for `I` is

```
Iw
```

Where `w` is the width of the integer data.

Example 14.13

Write a program to multiple data in a single statement.

```
      program read and write integers
       integer a,b,c
       write(*,101),'Enter numbers:'
       read(*,100),a,b,c
       write(*,101),a,b,c
100    Format(I3,I2,I3)
101    Format(I3,I5,I5)
       stop
       end
```

Explanation

If input is given as `12345678`, only first 3 integer digits (i.e. `123`) are stored in variable `a`, next two integer digits (`i.e. 45`) are stored in variable `b` and next three digits (`i.e. 678`) is stored in variable `c`. The output will be `123` for `a`, `45` for `b` and `678` for `c`. The format specification labelled 101 in above example shows the minimum number of characters (i.e field width) to be displayed for each data item. Here, format specification `I3` for variable `a` means the value of `a` (`i.e. 123`) should occupy at least three characters while displaying data in screen. Similarly, format specification `I5` for variable `b` means the value of variable `b` (`i.e. 45`) should occupy at least `5` characters while displaying in screen. The variable `b` has only two digits to display so, three extra spaces are added before its value to make minimum number of characters five.

If `n` integers have same width `w`, the single format specification can be written as `n*Iw` for `n` integers. This can be illustrated from following program.

Example 14.14

Write a program to read three integers of same width using I Format Specifier.

```
       program read and write integers
      integer a,b,c
      write(*,*),'Enter numbers:'
      read(*,100),a,b,c
      write(*,101),a,b,c
100   Format(3I3)
101   Format(3I5)
      stop
      end
```

Output

```
Enter numbers: 123456789 (say input)
123  456  789
```

Explanation:

The first three digits are stored in variable a, next three digits in b and last three digits in c.

The format description can be defined within `read()` or `write()` statement instead of separate format statement.

ii. F Format

The symbol F is used to denote the real data expressed in decimal form. The general form of the F format is

```
Fw.d
```

Where

w is field width and d is precision (i.e. number of digits to be displayed after decimal point)

Example 14.15

Write a program to read two fractional number and find their sum with precision 2.

```
      program read and write fractional numbers
      real a,b,c
      write(*,*),'Enter two numbers:'
      read(*,100),a,b
      c=a+b
      write(*,101),'a=',a,'b=',b,'Sum is:',c
100   Format(F6.3,F6.3)
101   Format(A3,F6.3,A3,F6.3,A10,F5.2)
      stop
      end
```

```
Output
Enter two numbers:
12.34565.789 (say input)
a=12.345 b=65.789   Sum is:78.13
```

iii. E Format

The symbol E is used to denote a real number expressed in exponential form. It's general form is:

```
E w.d
```

Where, w is the total field width including mantissa (i.e base), the letter E, the exponent and the sign of the exponent. And d is the decimal width (i.e. precision) of the mantissa.

Example 14.16

Read two real numbers in fractional form and display their sum in exponential form.

```
      program read and write real numbers
      real a,b,c
      write(*,*),'Enter First Number in fractional form:'
      read(*,100),a
      write(*,*),'Enter Second Number in fractional form:'
      read(*,100),b
      c=a+b
      write(*,101),'a=',a,'b=',b,'Sum is:',c
100   Format(F6.3)
101   Format(A3,F6.3,A3,F6.3,A10,E10.2)
      stop
      end
```

```
Output
Enter First Number in fractional form: 34.567
Enter Second Number in fractional form: 89.789
a=34.567 b=89.789   Sum is:  0.12E+03
```

iv. X Format

The symbol X is used to skip some columns in a data. Its general form is:

```
nX
```

Where n is the number of columns to be skipped in case of reading data from keyboard and number of blank spaces in case of outputting data in screen.

Example 14.17

Write a program to illustrate the use of X in format specification.

```
      program skip columns
      real a,b,c
      write(*,*),'Enter two numbers:'
      read(*,100),a,b
      c=a+b
      write(*,101),'a=',a,'b=',b,'Sum is:',c
100   Format(F6.3,1X,F6.3)
101   Format(A3,F6.3,4X,A3,F6.3,4X,A10,F5.2)
      stop
      end
```

```
Output
Enter two numbers:
12.345 78.567
a=12.345 b=78.567
Sum is:90.91
```

In above example, two numbers are given/input by separating space. The space is ignored or skipped at time of input. Similarly, the 4 extra blank spaces are added while outputting the data due to format specification 4X. If the format specification 1X is not given in format for reading, then the input data must be without space.

v. A Format

The symbol A is used for character type data. The general form is:

```
Aw
```

Where w is the width. The width is optional. If w is not specified in format specification, then the size of the variable declared in the character statement is considered as the width.

Example 14.18

Write a program to illustrate the use of A in format specification.

```
      program to read name
      character name*20
      write(*,*),'Enter your name:'
      read(*,900),name
      write(*,901),'Your Name is:',name
900   Format(A20)
901   Format(A15,A10)
      stop
      end
```

```
Output
Enter your name: Shyam Bahadur Thapa
Your Name is:Shyam Baha
```

Explanation:

Here, only 10 characters of name i.e. Shyam Baha is displayed due to format specification A10 in output format statement.

vi. T Format

The symbol T is used only in output format specification to denote the column from which the output must start. The general form is:

Tn

Where n specifies the number of column from which the output will be started.

Example 14.19

Write a program to illustrate the use of T in format specification.

```
      program tab format
      integer a,b
      a=10
      b=30
      write(*,901),'a=',a,'b=',b
901   Format(A2,T10,I2,A3,T20,I2)
      stop
      end
```

```
Output
a=        10 b=        30
Here, the value 10 starts from 10th
columns and 30 from 20th columns.
```

vii. Quote Format

The quote format specification is used in output format statement to print as it is.

Example 14.20

Write a program to illustrate the use of Quote format specification.

```
      program area of a cirlce
      real r,a
      write(*,*),'Enter radius:'
      read(*,900),r
      a=3.14*r*r
      write(*,901),r,a
900   Format(F5.2)
901   Format ('The area of circle having radius ',F5.2,' is:',F10.1)
      stop
      end
```

```
Output
Enter radius: 12.75
The area of circle having radius 12.75 is:      510.4
```

14.9 Control Statements

The statements which alter the flow of execution of the program are known as control statements. In the absence of control statements, the instructions or statements are executed in the same order in which they appear in the program. Such type of construct is known as sequential construct. However, in practice, sometimes we have to perform certain tasks depending upon the outcome of a logical test. We have to do certain calculation/task depending on whether a condition or test is true or false. Similarly, sometimes it is necessary to perform repeated actions or skip some statements. For these operations, control statements

are needed. They control the flow of program so that it is not necessary to execute statements in the same order in which they appear in the program. The control statement may be either decision making or loop.

Decision Making Statements

The decision making statement decides to execute certain statements or not. As they control the flow of execution; they also fall under the category of control statements. Using with goto statement, they can be used as loop also. The decision making types of control statements are classified as following:

i. Unconditional GO TO Statement
ii. Computed GO TO Statement
iii. Arithmetic If statement
iv. Logical If statement
v. If-then-else statement

Loops or Repeating Construct

Loop may be defined as block of statements which are repeatedly executed for a certain number of times or until a particular condition is satisfied. When an identical task is to be performed for a number of times, then loop is used. A loop is executed repeatedly till an expression is true. When the expression becomes false, the loop terminates and the control passes on to the statement following the loop. The FORTRAN 77 has only one loop: `do loop`, supported by ANSII standard. But most of compilers supports following types of loops.

i. Do Loop
ii. Do-While Loop
iii. Do—Until Loop

14.9.1 Unconditional GO TO Statements

The unconditional GO TO statement transfers program control to the statement identified by the statement label. It is called unconditional as the program control is transferred to any number statement in the program without checking any condition (i.e. just specify the statement number or label to jump there).

Syntax

```
GO TO label_name
```

Here, `label_name` is a label of an executable statement appearing in the same program unit.

i.e.

```
      ......
      GO TO 400
      ........
400   statement
      ..............
```

Example 14.21

Program to read a number from user and determine whether it is positive or negative using goto statement.

```
      program  goto unconditional
       integer num
       write (*,*) 'Enter a number'
       read(*,*) num
       if(num .GT. 0) then
         go to 10
        else
         go to 12
       endif
       write(*,*) 'The number is positive'
10     stop
       write(*,*) 'The number is negative'
12     end
```

Output

```
Enter a number 9
The number is positive
```

Here, the syntax GO TO Label_name , GOTO Label_name and GOTOLabel_name are same.

Example 14.22

Write a program to act as a loop using goto statement

```
    program  goto unconditional
    integer i, count;
    i=1
    write (*,*) 'How many times do you want to repeat a loop?'
    read(*,*),count
500 if(i .GT. count) then
        goto 100
    else
        write(*,*),'Welcome to you',i
        i=i+1
        goto 500

    endif
100 write(*,*) 'This is end of the program.....Bye....'
    stop
    end
```

Output

```
How many times do you want to repeat a loop? 5
Welcome to you 1
Welcome to you 2
Welcome to you 3
Welcome to you 4
Welcome to you 5
This is end of the program.....Bye....
```

14.9.2 Computed GO TO Statements

The computed GO TO statement transfers program control to one of several statements specified depending on the value of an integer expression.

Syntax

```
GO TO (n1, n2, n3, ……nk), i
```

Where, i is an integer variable and n1, n2, n3 ….nk are statement number of an executable statement appearing in the same program unit as the computed GO TO statement. A computed GO TO statement evaluates the integer expression and then transfers program control to the specified statement number on the basis of value of integer variable i.

i.e.

if i=1, control goes to statement number n1
if i=2, control goes to statement number n2
if i=3, control goes to statement number n3

.

if i=k, control goes to statement number nk

It is important to note down that if i<1 or i>k, the statement is ignored and control transfers to next statement of goto statement.

Example 14.23

Write a program to illustrate the computed goto statement.

```
      program  goto computed
     integer n
     write (*,*) 'Enter your option:1 for statement number 6,
    +2 for statement number 8 & 3 for statement number 10 '
     read(*,*) n
     go to (6,8,10)n
6    write(*,*) 'Welcome in line number 6'
     stop
8    write(*,*) 'Welcome in line number 8'
     stop
10   write(*,*) 'Welcome in line number 10'
     end
```

Output

```
Enter your option:1 for statement number 6, 2 for statement number 8 &
3 for statement number 10
2
Welcome in line number 8
```

Example 14.24

Write a program to read sex code (i.e. 1 for male and 2 for female) from user and display corresponding sex using computed goto statement.

```
program to find sex
integer sexcode
```

```
      write (*,*),'Enter sex code: 1 for Male and 2 for Female'
      read(*,*),sexcode .
      go to (7,9),sexcode
      go to 10
      write(*,*) 'You are male'
7     stop
      write(*,*) 'You are female'
9     stop
      write(*,*) 'Invalid Option'
10    stop
      end
```

Output

```
Enter sex code: 1 for Male and 2 for Female 2
You are female
```

14.9.3 Logical IF statement

The logical `if` statement checks logical condition and transfers the program control accordingly.

Syntax:

```
If(condition)  then
     Statement1
     .....

else
     Statement2
     .....

endif
```

Here, the condition is checked first. `If` condition produces true value, the `statement1` is executed, otherwise `statement2` is executed. The `if -else if-----else` structure can be used like in C.

Example 14.25

Write a program to read a number from user and determine whether it is even or odd.

```
      program to determine even or odd
      integer n
      write (*,*),'Enter a number:'
      read(*,*),n
      if(mod(n,2) .eq. 0) then
        write(*,*) 'The number is even'
      else
        write(*,*) 'The number is odd'
      endif
      stop
      end
```

Output

```
Enter a number:90
The number is even
```

Example 14.26

Write a program to read a number from user and determine whether it is positive or negative or zero.

```
program to determine positive or negative
 integer n
 write (*,*),'Enter a number:'
 read(*,*),n
 if(n .gt. 0) then
   write(*,*) 'The number is positive'
 else if(n .lt.0) then
   write(*,*) 'The number is negative'
 else
    write(*,*),'It is zero'
 endif
 stop
 end
```

Output

```
 Enter a number:-9
 The number is negative
```

14.9.4 Arithmetic If Statement

The arithmetic if statement is used to transfer the program control to a particular statement in the program depending upon the value of an expression whether it is negative, zero or positive.

Syntax:

```
if(Expression), n1,n2,n3
```

Where, the `Expression` is a valid FORTRAN arithmetic expression enclosed within parenthesis and `n1, n2, n3` are statement number.

The `expression` is evaluated first and it is checked whether it is `-ve`, `zero` or `+ve`. If the value of expression is `-ve` value, the program control is transferred to statement number `n1`. Similarly, the program control is transferred to statement number `n2` and `n3` when the value of expression is `zero` and `+ve` value respectively.

Example 14.27

Write a program to read a number from user and determine whether it is positive or negative or zero using arithmetic if statement.

```
      program to determine positive or negative
      integer n
      write (*,*),'Enter a number:'
      read(*,*),n
      if(n)8,10,6
6     write(*,*) 'The number is positive'
      goto 11
8     write(*,*) 'The number is negative'
      goto 11
10    write(*,*) 'It is zero'
```

```
    stop
11  end
```

Output

```
Enter a number:1
The number is positive
```

Example 14.28

Write a program to find the sum of following series upto n terms

$y=x+x^2+x^3+.....x^n$

```
    program to  find sum of series
    real x,n,sum
    integer i
    write (*,*),'Enter value of x:'
    read(*,*),x
    write(*,*), 'How many terms do you want to add?'
    read(*,*),n
    sum=0
    i=0
900 i=i+1
    sum=sum+x** i
    if(i-n) 900,901,901
       write(*,902) sum
901
902 format(1X, 'Sum is:',F10.3)
    stop
    end
```

Output

```
Enter value of x:2
How many terms do you want to add?3
Sum is:    14.000
```

[Note: `x** i` means raise `x` to the power `i` (i.e. x^i)]

14.9.5 Do loop

The `DO loop` is used to repeat execution of some statements for pre-determined number of times. It is also known as a `controlled` loop. It uses control variable with its beginning & ending value and increment/decrement size (i.e. step-size). It is exactly same as `for` loop in C programming.

Syntax:

```
DO [label,] loop-control-variable = initial-value, final-value [,step-size]
    statement1
    statement2
    ...
    statementn
label CONTINUE
```

The `label` denotes statement number which represents final statement in the `DO` loop. This final statement may be any executable statement but is usually a `CONTINUE` statement. The `loop-control-variable` may be a variable of type `INTEGER`, `REAL` or `DOUBLE PRECISION`.

The initial-value, final-value and step-size controls the number of iterations. The loop-control-variable is initialized with initial-value provided in loop structure, it is increased by size step-size and it is increased/decreased until it reaches to final-value. When the loop-control reaches to final-value, the control is transferred to statement number specified by the label. Here, step-size and label are optional and default value for step-size is 1. The statement enddo can be used to determine the end of do loop instead of using label and continue statement.

The step-size may be positive or negative.

i. **step-size is positive**
The program checks to see if the final-value is greater than or equal to the current value of loop-control-variable. If it is, then the statement block within the is executed. At this point the loop-control-variable is incremented by the step-size and again checked against the final-value. As long as the value of the loop-control-variable is less than or equal to the final-value, the loop continues. Only when the value of the loop-control-variable exceeds the final-value does the loop finish and control pass to the statement following the end of the loop.

ii. step-size **is negative**
The program checks to see if the final-value is less than or equal to the current value of loop-control-variable. If it is, then the statement block within the loop is executed. At this point the loop-control-variable is decremented by the step-size and again checked against the final-value. As long as the value of the loop-control-variable is greater than or equal to the final-value, the loop continues. Only when the value of the loop-control-variable is less than the final-value does the loop finish and control pass to the statement following the end of the loop.

Example 14.29

Write a program to display welcome message ten times

```
      program do loop
      integer i
      do 100 i=1,10,1
         write (*,*),'Welcome', i
100   continue
      end
```

Output

```
Welcome 1
Welcome 2
Welcome 3
Welcome 4
Welcome 5
Welcome 6
Welcome 7
Welcome 8
Welcome 9
Welcome 10
```

Example 14.30

Write a program to display all even numbers from 50 to 60.

```
      program do loop
      integer i
      do 100 i=50,60,2
          write (*,*), i
100   continue
      end
```

Output

```
 50   52   54   56   58   60
```

Example 14.31

Write a program to display all multiples of 3 (i.e. exactly divisible by 3) from 50 to 30.

```
      program do loop
      integer i
      do 100 i=50,30,-1
          if(mod(i,3) .eq.0) then
            write (*,*), i
          endif
100   continue
      End
```

Output

```
 48   45   42   39   36   33   30
```

Example 14.32

Write a program to find factorial of a number. Read the number from user.

```
      program factorial
      integer f,i,n
      write(*,*),'Enter a number:'
      read(*,*),n
      f=1
      do 100 i=1,n,1
           f=f*i
100   continue
      write(*,*),'The factorial is:',f
      end
```

end do

Output

```
Enter a number:5
The factorial is: 120
```

Example 14.33

Write a program to find sum of first n natural number

```
      program do loop to find sum of first n natural number
      integer i, n, sum
      sum = 0
      write(*,*) 'Enter value of n:'
```

```
        read(*,*) n
        do i = 1, n
        sum = sum + i
        enddo
        write(*,*) 'sum =', sum
        end
```

Output

```
Enter value of n:10
sum = 55
```

[It is important to note down that the statement enddo can be used to determine the end of loop instead of label and continue statement in loop structure.]

Example 14.34

Write a program to read a number from user and check whether a number for prime or not.

```
      program prime number
       integer n,i
       write(*,*) 'Enter a number to be tested for prime:'
       read(*,*),n
       do i=2, n-1,1
          if(mod(n,i).eq.0) then
            goto 900
           endif
       enddo
900    if(n .eq. i) then
         write(*,*), 'The number is prime'
       else
          write(*,*), 'The number is not prime'
       endif
       stop
       end
```

Output

```
Enter a number to be tested for prime:23
The number is prime
```

14.9.6 Do While Loop

The do while loop is used to execute a set of statements repeatedly while the specified condition is true.

Syntax:

```
Do  while(conditional_expression)
  body
enddo
```

or

```
Do label  while(conditional_expression)
    body
Label     continue
```

The specified conditional_expression is evaluated first. If the value of the expression is true, the statements in body of loop are executed. If the value of the expression is false, control is transferred to the statement following the DO WHILE loop. The end of body of loop is identified by either enddo or label & continue statement. If label is omitted from loop structure, the loop must be terminated with an END DO statement.

A DO loop iterates a specified fixed number of times but sometimes it is necessary to loop based on some kind of testable criterion which is not based on the number of iterations. In such situation, a do-while loop is used. The do-while construct can be accomplished using IF and GOTO statements. Thus, FORTRAN does not have a formal do-while loop structure. Even though this syntax of do--while loop is accepted by many compilers, it is not ANSI FORTRAN 77 standard. It can be implemented using if statement as

```
label if (conditional_expression) then
      statements
      goto label
    endif
```

Example 14.35

Write a program to print natural numbers 1 to 10 using do—while loop

```
      program do while loop
      integer i
       i=0
       do while(i .Lt. 10)
            i=i+1
            write (*,*), i
       enddo
       end
```

Output

```
 1  2  3  4  5  6  7  8  9  10
```

Example 14.36

Modify above program to use if and goto structure

```
      program if structure
      integer i
      i=0
900   if(i .lt.10) then
           i=i+1
           write (*,*), i
      goto 900
      endif
      end
```

Output

Same as above program

Example 14.37

Write a program to calculate and print all the powers of two that are less than or equal to 100.

```
      program to find power of 2
      integer n
      n = 1
      do while (n .LE. 100)
      n = 2*n
      if(n.le.100) then
        write (*,*) n
      endif
      end do
      end
```

Output

```
      2  4 8 16 32 64
```

Example 14.38

Modify above program to use if and goto statement

```
      program do while using if and go to
      integer n
      n = 1
10    if (n .le. 100) then
         n = 2*n
          if(n.le.100) then
              write (*,*) n
            endif
      goto 10
      endif
      end
```

Output

Same as above

Example 14.39

Write a program to add two numbers and display their sum. The program must ask next two numbers and add till user wants. (i.e. program would terminate if user doesn't want to add another numbers).

```
      program add two number multiple times
      integer a,b,sum
      character option
      option='y'
      do 900 while(option .eq. 'y')
        write(*,*) 'Enter two numbers:'
        read(*,*),a,b
        sum=a+b
        write(*,*), 'The sum is:',sum
        write(*,*),'Do you want to add next two number? Enter y or n:'
        read(*,*),option
```

```
900   continue
      end
```

```
Output
Enter two numbers: 10 11
The sum is: 21
Do you want to add next two number? Enter y or n:
y
Enter two numbers: 50 4
The sum is: 54
Do you want to add next two number? Enter y or n:
n
```

14.9.7 Do Until Loop

If the termination criteria is at the end instead of the beginning of loop, it is often called an until-loop. In `until` loop, the body of the loop is executed first without testing condition. At the end of the loop, test condition is evaluated. If the condition is true, the program continues to evaluate the body of the loop once again. This process continues as long as the condition is true. When the condition becomes false, the loop is terminated, and the control goes to the statement that appears immediately after the body of until loop.

Syntax

```
do
      statements
until (conditional_expression)
```

This type of construct can be accomplished using `IF` and `GOTO` statements. Thus, FORTRAN does not have a formal `until` loop structure. Even though this syntax of `until` loop is accepted by many compilers, **it is not ANSI FORTRAN 77 standard**. It can be implemented using if statement as

```
label continue
      statements
if (conditional_expression) goto label
```

Example 14.40

Write a program to read a number until user enters zero. Finally count entered numbers and find their sum.

Method I: using do..until loop

```
      program implementation of until loop
      integer n, sum,count
      sum=0
      count=0
   do 900
      write(*,*) 'Enter a number:'
      read(*,*) n
      if(n .NE.0) then
          sum=sum+n
          count=count+1
```

```
        endif
        until(n .NE.0)
900   continue
        write(*,*) 'You entered',count,' positive numbers'
        write(*,*) 'The sum of numbers is:',sum
        end
```

Method II: using if and goto statement

```
        program implementation of until loop using if and goto
        integer n, sum,count
        sum=0
        count=0
100     continue
        write(*,*) 'Enter a number:'
        read(*,*) n

        if(n .NE.0) then
            sum=sum+n
            count=count+1
        endif
        if(n .NE.0) goto 100
        write(*,*) 'You entered',count,' positive numbers'
        write(*,*) 'The sum of numbers is:',sum
        end
```

```
Output
 Enter a number:20
 Enter a number:21
 Enter a number:1
 Enter a number:3
 Enter a number:4
 Enter a number:0
 You entered 5 positive numbers
 The sum of numbers is: 49
```

14.9.8 Nested Loop

When the body part of the loop contains another loop then the inner loop is said to be nested within the outer loop. Since a loop may contain other loops within its body, there is no limit on the number of loops that can be nested. In the case of nested loop, for each value or pass of outer loop, inner loop is completely executed. Thus, inner loop operates/moves fast and outer loop operates slowly.

Syntax:

```
do  counter_variable_for_outer=initial_value final_value step_size
    .
    .
  do  counter_variable_for_inner=initial_value final_value step_size
      statement1
      .
  enddo
enddo
```

Example 14.41

Write a program to display multiplication table of 5 -15

```
        program nested  loop
        integer i,j,product
        do i=5,15
        write(*,*) 'The multiplication table of ',i
```

```
      do j=1,10
       product=i*j
       write(*,*) product
       enddo
     enddo
     end
```

Output

```
The multiplication table of  5
 5  10  15  20  25  30  35  40  45  50
The multiplication table of  6
```

so on

Example 14.42

Write a program to read two numbers n1 and n2. Display all prime numbers between n1 and n2.

```
    program prime number
    integer n1,n2,i
    write(*,*) 'Enter lower limit n1 and upper limit n2:'
    read(*,*),n1,n2
    do i=n1, n2
    do j=2, i-1
       if(mod(i,j).eq.0) then
         goto 900
       endif
    enddo
900 if(i .eq. j) then
      write(*,*), i
    endif
    enddo
    stop
    end
```

Output

```
  Enter lower limit n1 and upper limit n2: 20 50
   23  29  31  37  41  43  47
```

14.10 Array

An array is a group of related data items that share a common name. In other words, an array is a data structure that store a number of data items as a single entity (object). The individual data items are called elements and all of them have same data types. Array is used when multiple data items that have common characteristics are required. In other words, an array is a collection of individual data elements that is ordered, fixed in size and homogeneous. Suppose we have 20 numbers of type integer and we have to sort them in ascending or descending order. If we have no array, we have to define 20 different variables like a1, a2, a3,.....,a20 of type integer to store these twenty numbers which will be possible but inefficient. If the number of integers increases the number of variables will also be increased and defining different variables for different numbers will be impossible and inefficient. In such situation where we have multipl

data items of same type to be stored, we can use array. In array system, an array represents multiple data items but they share same name. The individual elements are characterized by array name followed by one or more subscripts or indices enclosed in small brackets.

14.10.1 One Dimensional Array

We can define one dimensional array in two ways:

First Method

```
array_name(array_length)
```

Here, the *array_name* is name of the array and the *array_length* is the number of elements in the array. In this instance, the elements of the array have the subscripts 1, 2, 3, ..., array_length. The array index starts with one and ends with array_length unlike C.

Second Method

```
array_name(lower_bound:upper_bound)
```

Here *lower_bound* and *upper_bound* must be INTEGER constants and their values may be negative, zero or positive. The array index starts with *lower_bound* ends with *upper_bound*. Thus, we can define an array such that its initial index starts different than one.

Example 14.43

Write a program to illustrate the process of reading and displaying an array.

```
program array
integer num(5),i
write(*,*) 'Enter five integers for an array:'
do i=1,5
   read(*,*),num(i)
enddo
write(*,*) 'The array elements are:'
   do i=1,5
   write(*,*),num(i)
enddo
stop
end
```

Output

```
Enter five integers for an array: 10  34  56  7  89
 The array elements are: 10 34  56  7  89
```

Example 14.44

Modify above program to illustrate the process of array declaration with lower bound and upper bound.

```
program array
 integer num(6:10),i
write(*,*) 'Enter five integers for an array:'
do i=6,10
```

```
    read(*,*),num(i)
enddo
write(*,*) 'The array elements are:'
   do i=6,10
   write(*,*),num(i)
enddo
stop
end
```

Output

```
Same as above program
```

In this example, the array size is again 5 but its index begins with 6 and ends with 10.

Initialization of an array

i. The DATA statement

The DATA statement is a statement used to initialize variables.

Syntax:

```
DATA variable_list/constant_list/ [,variable_list/constant_list/] ...
```

Each `variable_list` is a list of variables, and each `constant_list` a list of constants, separated by commas in each case. Each `constant_list` must contain the same number of items as the preceding `variable_list` and corresponding items in sequence in the two lists must be of the same type. The DATA statement assigns to each variable in each `variable_list` a value equal to the corresponding constant in the corresponding `constant_list`.

Example 14.45

Write a program to initialize variables with DATA statement

```
program variable initialization
integer a,b,c
DATA a,b,c/10,34,50/
write(*,*) 'The values of variables:'
write(*,*),'a=',a,' b=',b,' c=',c
stop
end
```

Output

```
The values of variables:
 a= 10 b= 34 c= 50
```

Example 14.46

Write a program to initialize array with data statement.

```
 program array
integer num(5),i
data num(1),num(2),num(3),num(4),num(5)/100,200,300,900,500/
write(*,*) 'The array elements are:'
   do i=1,5
   write(*,*),num(i)
```

```
        enddo
        stop
        end
```

Output

```
The array elements are: 100  200  300  900  500
```

ii. Implied DO list

When a large number of array elements have to be initialized, we can use an implied DO list instead of individual elements. An implied DO list is used in a DATA statement or an input/output statement to generate a list of array elements. The simplest form of implied DO list is:

```
(array_element, counter_variable=initial_value, final_value [,step_size])
```

where array_element is a list of array elements separated by commas. The counter_variable in implied DO is defined only in the implied DO list, and is distinct from any variable of the same name used elsewhere.

Example 14.47

Write a program to initialize array with implied do list and data statement

The following part of program initializes all elements of array num with 10.

```
integer num(5),i
data(num(i),i=1,5)/5*10/
```

To initialize array with different values, we can use following statement.

```
data(num(i),i=1,5)/10,80,60,20,30/
```

Example 14.48

Write a program to read and display an array elements using implied do structure.

```
program array
integer num(5),i
write(*,*) 'Enter five elements of an array:'
read(*,*),(num(i),i=1,5)
write(*,*) 'The array elements are:'
write(*,*),(num(i),i=1,5)
stop
end
```

Output

```
Enter five elements of an
12,34,54,33,24
 The array elements are:
 12  34  54  33  24
```

Here,

```
read(*,*),(num(i),i=1,5)
```

is equivalent to

```
do i=1,5
   read(*,*),num(i)
enddo
```

Example 14.49

Write a program to read 5 integers and calculate average & deviation of each number from average value

```
      program array
      integer num(5),i,sum
      real avg
      sum=0
      do i=1,5
       write(*,*)'Enter a number:'
       read(*,*) num(i)
       sum=sum+num(i)
      enddo
      avg=sum/5
      write(*,*) 'The average value is:',avg
      write(*,*) 'The deviation from each number from average is:'
      do i=1,5
          write(*,*) 'number:',num(i), ' deviation:',num(i)-avg
      enddo
      end
```

Output

```
 Enter a number:10
 Enter a number:11
 Enter a number:12
 Enter a number:13
 Enter a number:14
 The average value is:  12.
 The deviation from each number from average is:
 number: 10 deviation: -2.
 number: 11 deviation: -1.
 number: 12 deviation:  0.
 number: 13 deviation:  1.
 number: 14 deviation:  2.
```

Example 14.50

Write a program to display following pattern upto n lines

```
*
**
***
****
*****
```

```
    program pattern
    integer n,i,j
    write(*,*) ,'Enter a number:'
     read(*,*),n
     do i=1,n
        write(*,*),('*',j=1,i)
     enddo
     stop
     end
```

Example 14.51

Write a program to display following pattern up to n lines

```
1
2 2
3 3 3
4 4 4 4
5 5 5 5 5
```

```
program pattern
 integer n,i,j
 k=1
 write(*,*) ,'Enter a number:'
 read(*,*),n
 do i=1,n
     write(*,*),(i,j=1,i)
enddo
stop
end
```

Example 14.52

Write a program to find maximum number among five numbers

```
 program array
integer num(5),i,max
do i=1,5
 write(*,*)'Enter a number:'
 read(*,*) num(i)
 enddo
 max=num(1)

do i=1,5
 if(max.LT.num(i)) then
    max=num(i)
 endif
enddo
write(*,*) 'The maximum number is:',max
end
```

```
Output
Enter a number:2
Enter a number:34
Enter a number:1
Enter a number:45
Enter a number:6
The maximum number is: 45
```

Example 14.53

Write a program to read 10 integers from user and sort them in ascending order. Finally display the sorted numbers.

```
program sorting
integer num(10),i,j,temp
write(*,*)'Enter 10 numbers:'
read(*,*) (num(i),i=1,10)

do i=1,9
    do j=i+1,10
    if(num(i) .gt. num(j)) then
```

```
Output
Enter 10 numbers:
10,22,446,4,3,26,90,345,7,90
The numbers in sorted order:
4 7 10 22 26 90 90 345 446
```

```
            temp=num(i)
            num(i)=num(j)
            num(j)=temp
        endif
        enddo
    enddo

    write(*,*) 'The numbers in sorted order:'
    write(*,*)(num(i),i=1,10)
    end
```

Example 14.54

Write a program to read marks of 10 students in a subject and display top three marks.

```
program sorting
integer marks(10),i,j,temp
write(*,*)'Enter marks of ten students:'
read(*,*) (marks(i),i=1,10)
do i=1,9
    do j=i+1,10
    if(marks(i) .lt. marks(j)) then
        temp=marks(i)
        marks(i)=marks(j)
        marks(j)=temp
    endif
    enddo
enddo
write(*,*) 'The top three marks:'
write(*,*)(marks(i),i=1,3)
end
```

```
Output
Enter marks of ten students:
89,78,45,67,34,98,38,87,54,36
The top three marks: 98 89 87
```

Example 14.55

Write a program to read marks of 10 students in a subject and display second highest marks.

```
program sorting
integer marks(10),i,j,temp
write(*,*)'Enter marks of ten students:'
read(*,*) (marks(i),i=1,10)
do i=1,9
    do j=i+1,10
    if(marks(i) .lt. marks(j)) then
        temp=marks(i)
        marks(i)=marks(j)
        marks(j)=temp
    endif
    enddo
enddo
write(*,*) 'The second highest mark is:'
write(*,*)marks(2)
end
```

```
Output
Enter marks of ten students:
23, 45, 67, 89, 13, 24, 47,
86, 54, 32
The second highest mark is: 86
```

14.10.2 Multi-dimensional Array

The multi-dimensional array is array with multiple subscripts. FORTRAN77 allows us to have up to seven subscripts for a single array. The number of subscripts is referred to as the DIMENSION of the array. The multi-dimensional array is defined as

Syntax:

```
datatype array_name(dimension1, dimension2,.....)
```

Examples:

```
Integers matrix(2,4)
Real table(2,3,4)
```

Here, `matix` is 2-dimensional array while `table` is three dimensional array.

Array Representation in memory

Unlike C programming languages, FORTRAN stores arrays in column-major form. The left-most array index varies the most rapidly, followed by the next left-most array index, etc. Thus the last index in an array varies the most slowly. Let us consider an array `B(2,4)`: the elements are stored in memory as following

The matrix

```
B11    B12    B13    B14
B21    B22    B23    B24
```

is stored as

```
B11    B21    B12    B22    B13    B23    B14    B24
```

Declaration of 2-D array

Syntax:

```
Datatype array_name(dimension1, dimension2)
e.g. real mat(4,5)
```

Alternative Method

```
Datatype array_name(lower_limit_for_dim1: upper_limit_for_dim1,
                    lower_limit_for_dim2: upper_limit_for_dim2)
```

e.g.

`real mat(5:6, 10:15)` is equivalent `real mat(2,6)`

Example 14.56

Write a program to read matrix of order 2*3 and display in matrix form.

```
program two dim array
integer matrix(2,3),i,j
do i=1,2
  do j=1,3
```

77 allows us to have
s referred to as the

jor form. The left-
y index, etc. Thus,
2, 4) : the elements

```
      write(*,*)'Enter a matrix(',i,',',j,'):'
      read(*,*) matrix(i,j)
    enddo
enddo
write(*,*)'In Matrix Form:'
do i=1,2
    write(*,*)(matrix(i,j),j=1,3)
enddo
end
```

Output

```
Enter a matrix( 1, 1):1
Enter a matrix( 1, 2):2
Enter a matrix( 1, 3):3
Enter a matrix( 2, 1):4
Enter a matrix( 2, 2):5
Enter a matrix( 2, 3):6
In Matrix Form:
1 2 3
4 5 6
```

Example 14.57

Write a program to read two matrices of same size and display their sum matrix.

```
 program matirx addition
integer mat1(2,3),mat2(2,3),result(2,3),i,j
write(*,*) 'Enter first matirx of size 2*3:'
do i=1,2
   read(*,*)(mat1(i,j),j=1,3)
enddo
write(*,*) 'Enter second matirx of size 2*3:'
do i=1,2
   read(*,*)(mat2(i,j),j=1,3)
enddo
do i=1,2
   do j=1,3
     result(i,j)=mat1(i,j)+mat2(i,j)
   enddo
enddo
 write(*,*) 'The sum matrix is:'
do i=1,2
    write(*,*)(result(i,j),j=1,3)
enddo
end
```

Output

```
Enter first matirx of size 2*3: 1 2 3 4 5 6
Enter second matirx of size 2*3: 9 8 7 6 5 4
The sum matrix is:
 10 10 10
 10 10 10
```

Example 14.58

Write a program to read a matrix of size 2*3 and multiply each element by 3.

```
program matirx
integer mat(2,3),result(2,3),i,j
write(*,*) 'Enter  matirx of size 2*3:'
do i=1,2
   read(*,*)(mat(i,j),j=1,3)
enddo
do i=1,2
   do j=1,3
    result(i,j)=mat(i,j)*3
enddo
enddo
write(*,*) 'The result matrix is:'
do i=1,2
     write(*,*)(result(i,j),j=1,3)
enddo
end
```

```
Output
Enter  matirx of size 2*3:
1
2
3
4
5
6
 The result matrix is:
 3      6      9
 12     15     18
```

Example 14.59

Write a program to read a matrix and find its transpose matrix

```
 program matirx
integer mat(2,3),result(3,2),i,j
write(*,*) 'Enter  matirx of size 2*3:'
do i=1,2
   read(*,*)(mat(i,j),j=1,3)
enddo
do i=1,2
 do j=1,3
    result(j,i)=mat(i,j)
enddo
enddo
write(*,*) 'The original matrix is:'
do i=1,2
     write(*,*)(mat(i,j),j=1,3)
enddo
write(*,*) 'The transpose matrix is:'
do i=1,3
     write(*,*)(result(i,j),j=1,2)
enddo
end
```

```
Output
Enter  matirx of size 2*3:
1 2 3
4 5 6
The original matrix is:
1 2 3
4 5 6
The transpose matrix is:
1 4
2 5
3 6
```

Example 14.60

Write a program to read two matrices of suitable order for multiplication. Find product matrix.

```
 program matirx multiplication
INTEGER  m,n,p,q
PARAMETER(m=2,n=3,p=3,q=4)
```

```
integer mat1(m,n),mat2(p,q),result(m,q),i,j,k,psum
write(*,*) 'Enter first matirx of size 2*3:'
do i=1,m
   read(*,*)(mat1(i,j),j=1,n)
enddo

write(*,*) 'Enter second matirx of size 3*4:'
do i=1,p
   read(*,*)(mat2(i,j),j=1,q)
enddo

psum=0
do i=1,m
   do j=1,q
      do k=1,n
      psum=psum+(mat1(i,k)*mat2(k,j))
      enddo
      result(i,j)=psum
      psum=0
   enddo
enddo
write(*,*) 'The product matrix is:'
do i=1,m
      write(*,*)(result(i,j),j=1,q)
enddo
end
```