UNIT-2

PROGRAMMING METHODOLOGY

Stylistic Guidelines:

Writing good program is a skill. This can be developed by using the following guidelines .

1. **Meaningful Names for identifiers:** A programmer should give the meaningful names to each section of the program so that it can help him to identify the variable used for specific purpose. This helps him to execute the right elements during the complex run of a program.
2. **Ensure clarity of expression**: Expression carry out the specified action. Thus they must be clearly understood by the users. There should not be any compromise with the clarity of expression.
3. **Use of comments and indentations**: Comments generally are used as the internal documentation of a program .if comments are used in the program they will guide the program while debugging and checking. While indentation is the proper way of writing to avoid the confusion regarding the flow of program. These highlights nesting of groups of control statements.
4. **Insert blank lines and blank spaces**: Blank lines should be used to separate long, logically related blocks of code. Specifically Normally in programming the standard for the use of spaces is to follow normal English rules. This means that: Most basic symbols in C++ (e.g., "=", "+", etc.) should have at least one space before and one space after them.
5. **Statements**: Each statement should appear on a separate line. The opening brace following a control statement such as if or while should appear on the line after the if or while, lined up with the left of the control statement, and the closing brace should appear on its own line, lined up with the left of the control statement. The opening and closing braces for a function should be lined up in the same way. The statements within a {_____} pair are indented relative to the braces.

Characteristics of a Good Program:

Following are the characteristics of a good program.

1. **Effective and efficient:** The program produces correct results and is faster, taking into account the memory constraints.
2. **User friendly:** The program should be user friendly. The user should not be confused during the program execution . The user should get correct direction and alerts when he is going through the program.
3. **Self documenting code:** A good program must have self documenting code. This code will help the programmer to identify the part of the source code and clarify their meaning in the program.
4. **Reliable:** The good program should be able to cope up from any unexpected situations like wrong data or no data.
5. **Portable:** The program should be able to run on any platform, this property eases the use of program in different situations.
6. **Robustness**: Robustness is the ability of the program to bounce back an error and to continue operating within its environment

PROBLEM SOLVING METHODOLOGY AND TECHNIQUES:

To develop an efficient and effective programs we should adopt a proper problem solving methodology and use appropriate techniques. Following are some of the methods and techniques to develop a good program.

1. **Understand the problem well**: for a good program one should understand the problem well . one should know what exactly is expected from the problem. Knowing the problem well is the half way done.
2. **Analyze the program. :** Analyzing the problem involves identifying the program specification and defining each program's minimum number of inputs required for output and processing components.
3. **Design** : In this phase of design a Model is developed which look alike a program . This phase gives the face to the program. Outputs are designed in this phase.

4. **Code program :**This step is the actual implementation of the program. In this program algorithm is translated into programming language. in this it is essential to decide which technique or logical will be more appropriate for coding.
5. **Test and Debug program.:** Once the solution algorithm is coded the next step is to test and debug the program. Testing is the process of finding errors in a program and debugging is of correcting the errors. The developed program is put to test in different conditions and verified at different level for its proper and efficient working.
6. **Implementation & Documentation:** After successful execution of the program it is implemented. Documentation refers to written descriptions specification, design code and comments, internal and external to program which makes more readable and understandable.

Uses of documentation:

1. This becomes an useful interface between a technical personnel and non technical personnel.
2. This is very useful for upkeep and maintenance.
3. Documentation makes ease for any technical emergencies.
4. Very useful in operating for learners and trainers.

Algorithm :-Algorithm is a step-by-step process of solving a well-defined computational problem. An algorithm is an effective method expressed as a finite list of well-defined instructions for calculating a function, starting from an initial state and initial input. Thus, a step-by-step procedure to solve the problem is called algorithm.

Example: Let us take one simple day-to-day example by writing algorithm for making „Maggi Noodles" as a food.

Step 1: Start
Step 2: Take pan with water
Step 3: Put pan on the burner
Step 4: Switch on the gas/burner
Step 5: Put maggi and masala
Step 6: Give two minutes to boil
Step 7: Take off the pan
Step 8: Take out the magi with the help of fork/spoon
Step 9: Put the maggi on the plate and serve it
Step 10: Stop.

Further, the way of execution of the program shall be categorized into three ways: (i) sequence statements; (ii) selection statements; and (iii) iteration or looping statements. This is also called as „control structure".

Sequence statements: In this program**,** all the instructions are executed one after another.
Example : Write an algorithm to print „Good Morning‟.
Step 1: Start
Step 2: Print „Good Morning‟
Step 3: Stop
Example: Write an algorithm to find area of a rectangle.
Step 1: Start
Step 2: Take length and breadth and store them as L and B?
Step 3: Multiply by L and B and store it in area
Step 4: Print area
Step 5: Stop

In the above mentioned two examples (Example II and III), all the instructions are executed one after another. These examples are executed under sequential statement. **Selective Statements:** In this program, some portion of the program is executed based upon the conditional test. If the conditional test is true, compiler will execute some part of the program, otherwise it will execute the other part of the program.

Example : Write an algorithm to check whether a person is eligible to vote? (Age more than or equal to 18 years makes one eligible to vote).
Step 1: Start
Step 2: Take age and store it in age
Step 3: Check age value, if age >= 18 then go to step 4 else step 5
Step 4: Print "Eligible to vote" and go to step 6
Step 5: Print "Not eligible to vote"
Step 6: Stop

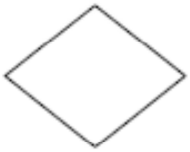
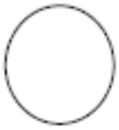
Example : Write an algorithm to check whether given number is +ve, -ve or zero.
Step 1: Start
Step 2: Take any number and store it in n.
Step 3: Check n value, if n > 0 then go to step 5 else go to step 4
Step 4: Check n value, if n < 0 then go to step 6 else go to step 7
Step 5: Print "Given number is +ve" and go to step 8
Step 6: Print "Given number is -ve" and go to step 8
Step 7: Print "Given number is zero"
Step 8: Stop
In the above mentioned examples IV and V, all the statements are not executed, but based upon the input, some portions of the algorithm are executed, because we have „true‟ or „false‟ situation in the program.

Iterative statements: In some programs, certain set of statements are executed again and again based upon conditional test. i.e. executed more than one time. This type of execution is called „looping or iteration‟.
Example: Write an algorithm to print all natural numbers up to „n‟.
Step 1: Start
Step 2: Take any number and store it in n.
Step 3: Store 1 in I
Step 4: Check I value, if I<=n then go to step 5 else go to step 8
Step 5: Print I
Step 6: Increment I value by 1
Step 5: Go to step 4
Step 8: Stop

In the above example, steps 4, 5, 6 and 7 are executed more than one time.

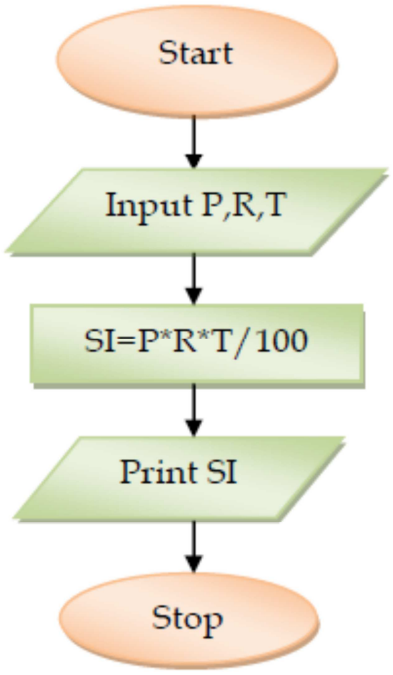
Flowchart :-We can also show steps of an algorithm in graphical form by using some symbols. This is called flowcharting. Thus, Flowchart is a pictorial view of algorithm.

Flowchart Symbols:-Some of the standard symbols along with respective function(s) that are used for making flowchart are as follows:

Symbols	Functions
1. 	Start/stop
2. 	Input/output
3. 	Processing
4. 	Decision Box
5.	Flow of control
6. 	Connector

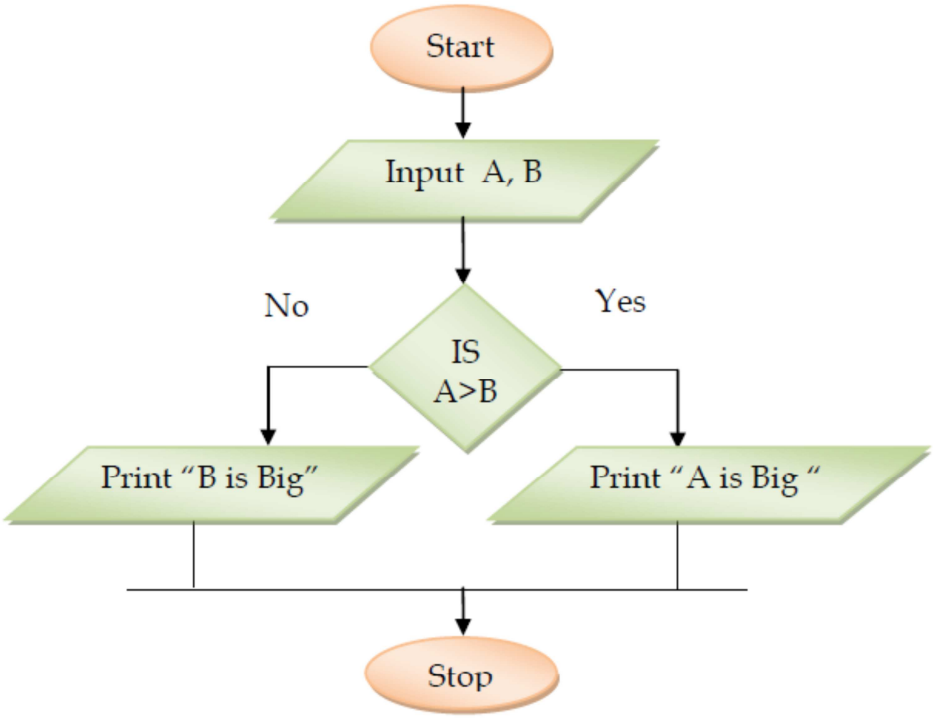
The following flowchart is an example of a sequential execution.

Example : Draw a flowchart to find the simple interest. (Sequence)
Solution:

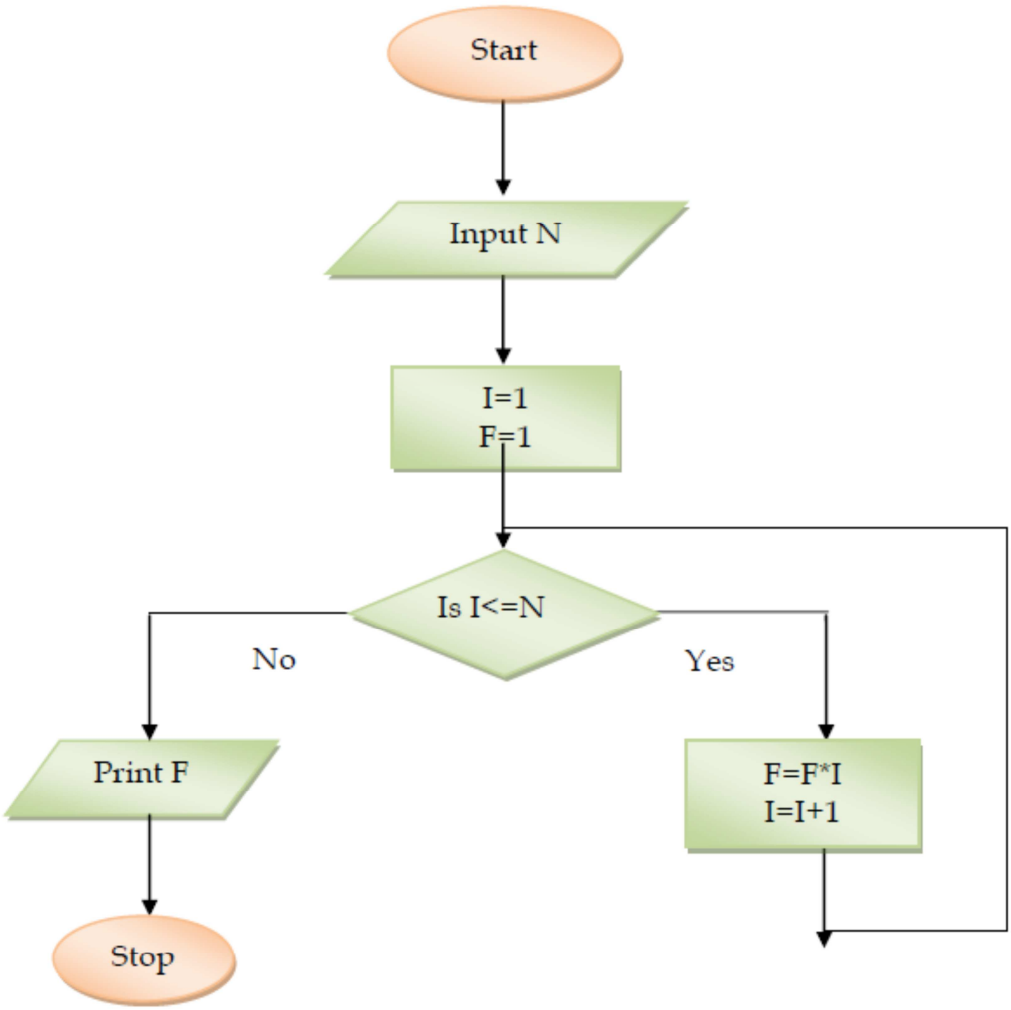


The following flowchart is an example of a selective execution.

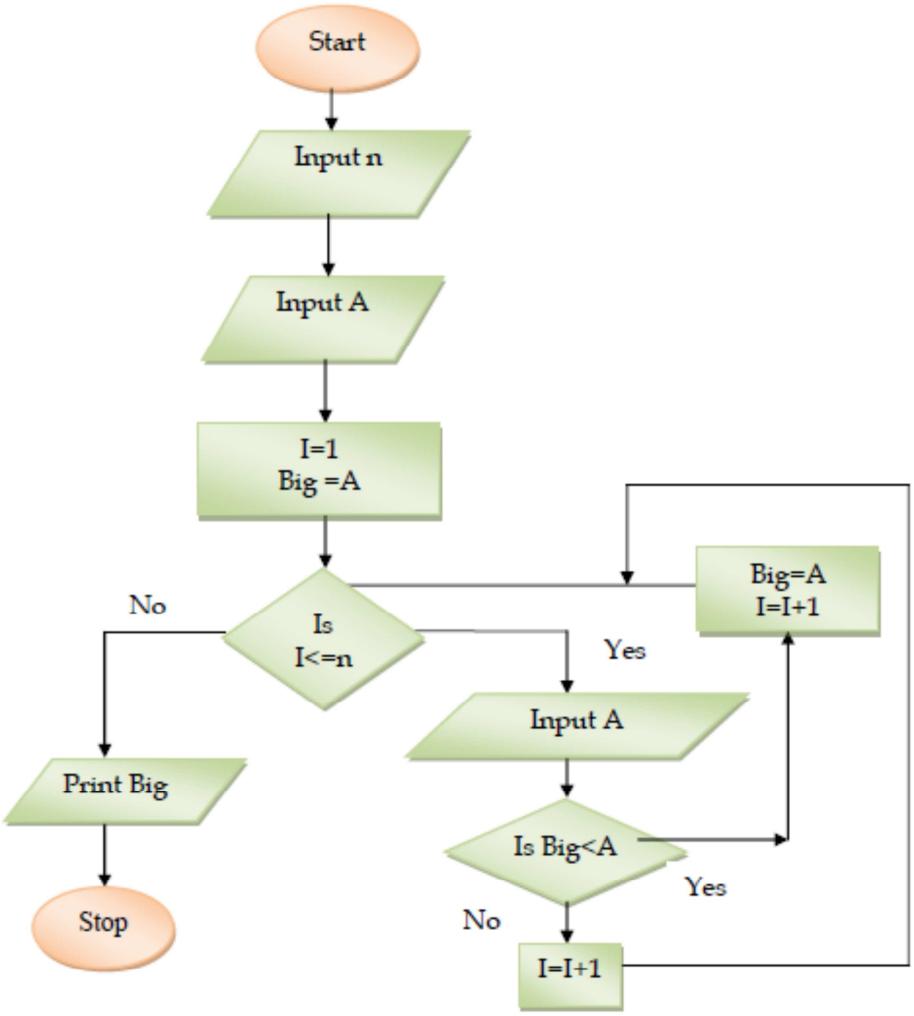
Example : Draw a flowchart to find bigger number among two numbers (selective)
Solution:



Following are the examples of an iterative execution.
Example : Draw a flow chart to find factorial of any number.
Solution:



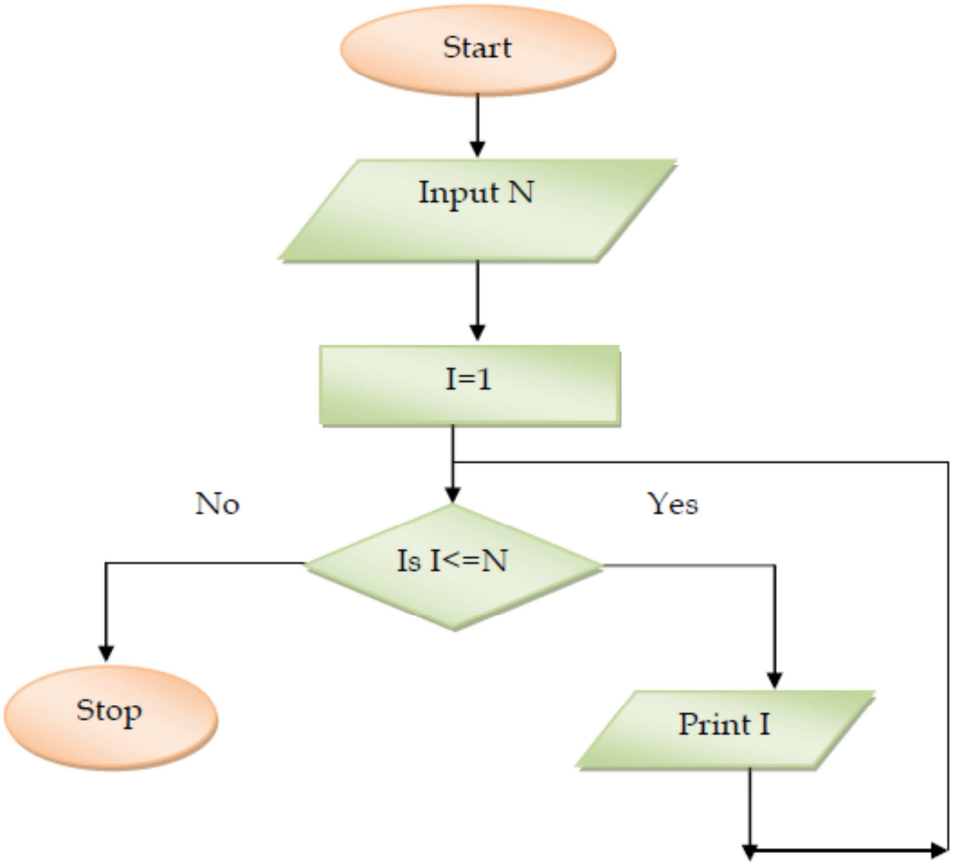
Example: Draw a flow chart to find biggest number among "n" numbers.
Solution:



Finite and Infinite loop: In looping statements, if some set of statements are executed n times (fixed number of times), then it is called finite loop. At the same time, if some set of statements are executed again and again without any end (infinite times), then it is called infinite loop . For example (X), if we are not incrementing I (index) value, then we will get endless (infinite) loop. Set of statements is executed again and again without any end is called infinite loop. The following is an example of infinite

Example : Draw a flow chart to describe an infinite loop.

Solution:



In the above example value of I is not incremented, so it will create endless loop.
This is also called infinite loop.

Type of errors: There are three types of errors generally occur during compilation and running a program. They are (i) Syntax error; (ii) Logical error; and (iii) Runtime error.

Syntax error: Every programming language has its own rules and regulations (syntax). If we overcome the particular language rules and regulations, the syntax error will appear (i.e. an error of language resulting from code that does not conform to the syntax of the programming language). It can be recognized during compilation time.

Example

```
int a = 0;
while (a < 10)
{
a = a + 1
cout<<a;
}
```

In the above statement, the fourth line is not correct. Since the statement does not end with **;**. This will flash a syntax error.

Logical error: Programmer makes errors while writing program that is called logical error. It is an error in a program's source code that results in incorrect or unexpected result. It is a type of runtime error that may simply produce the wrong output or may cause a program to crash while running. The logical error might only be noticed during runtime, because it is often hidden in the source code and are typically harder to find and debug.

```
int a = 100;
while (a < 10)
{
a = a + 1;
cout<< a;
}
```

In the above example, the while loop will not execute even a single time, because the initial value of a is 100.

Runtime error: A runtime error is an error that causes abnormal termination of program during run time. In general, the dividend is not a constant but might be a number typed by you at runtime. In this case, **division by zero** is illogical. Computers check for a "division by zero" error during program execution, so that you can get a "division by zero" error message at runtime, which will stop your program abnormally.