Introduction to Programming

Programming is the process of creating a set of instructions that tell a computer how to perform a task. Programming can be done using a variety of computer "languages," such as C, C++, SQL, Java and Python.

Language Classification

As the involvement of computer, automation and robotics growing in our daily life, programming becomes highly required to control all of them.
To control all of these systems and machines and take desired output by them skilled programming languages is necessary.
All the computer programming languages are broadly classified into the following;

- **Machine level Language**(low level)
- **Assembly level Language**(Middle level)
- **High-level Language**(High level)

Machine level languages (First Generation of programming language):

It's the lowest level and named as first generation of programming language. Machine level language consist only two condition i.e. either true (1) or false (0); this type of language known as binary language.

A computer system could understand only binary language i.e. all the instruction feed into the computer system must be in the form of 0 or 1. Machine level languages are very tough to understand by the humans.

Advantages of machine level language:

- Machine level languages are directly interacting with computer system.
- There is no requirement of software of conversion like compiler or interpreters.
- It takes very less time to execute a program, because there is no conversion take place.

Disadvantages of machine language:

- Its machine dependent language i.e. individual program required for each machine.
- To develop a program in machine language, it's too hard to understand and program.
- Its time consuming to develop new programs.
- Debugging process is very hard because of finding errors process is typical.
- Machine language is not portable language.

Assembly level languages (Second Generation programming language):

It's a middle level and named as second generation programming language. It contains the same instruction as machine level language, but the instructions and the variables have specific name or called commands instead of being just binary numbers.

For example;

```
LOAD          BASEPAY
ADD           OVERPAY
STORE         GROSSPAY
```

Advantages of Assembly language:

- It is easily understood by human because it is uses statements instead of binary digits.
- To develop a program it takes less time.
- Debugging and troubleshoot is easy due to easily find error.
- It's a portable language.

Disadvantages of Assembly language:

- Sometime it's hard to understand the statement or command use.

High-level language (Third Generation):

High level language is the upper level language and also known as third generation programming language. It does consider as high level because, which language comes under this category are closer to human languages. Hence this is highly understood programming language by human. There have many examples of high level languages such as, FORTRAN, Pascal, C, C++, JAVA, ADA, COBOL, LISP, Prolog etc.
The first high level programming language was written in 1950s. Those programs written in high level language must require software or a set of program to translate that program into machine understandable. This software called compiler and/or interpreter. The main job of compiler and translator is to take the source code of the program and convert that code into the machine understood code.

Advantages of high level language:

- In this instructions and commands much easier to remember by programmer.
- Its logic and structure are much easier to understand.
- Debugging is easier compare to other languages.
- Less time consuming to writing new programs.
- HLL are described as being portable language.

Disadvantages of high level language:

- HLL programming language take more space compare to other MLL (machine level language) and/or ALL (Assembly level language).
- This programming language execute slowly.

Problem Solving Using Computers

Computers are used in modern life in a number of ways. They are used to store data for us, to solve problems for us, to play music and videos for us, to communicate with our friends and so on, the list is endless. A stage is reached where people tend to call it as **"The Electronic Brains"**.

Problem solving can be defined as "**The task of expressing the solution of complex problems in terms of simple operations understood by the computer**".
Problem solving using a computer will thus require us to follow a well-defined sequence of steps in a systematic manner. In order to solve a problem using a computer we will have to pass through certain stages. The stages are:
1. Problem definition
2. Problem analysis
3. Design of a solution using design tools such as Flowcharts and Algorithms
4. Programming the computer (Coding)
5. Checking and correcting errors (Testing and Debugging)
6. The documentation of the program
7. Program enhancement or maintenance

PROBLEM DEFINITION
The basic concept, which should be clearly understood, is that "**Success can be achieved in problem solving only when we know what we want to do**", i.e., the problem should be clearly understood first by the user. A proper involvement in this process always helps in generating a good workable solution. We cannot make any progress if we do not understand the problem.

PROBLEM ANALYSIS
In problem analysis we try to understand the requirements of the problem to be solved. This process is the first step towards the solution domain. Explicit requirements about the input and output, the time constraints, the processing requirements, accuracy, memory limitations, error handling and interfaces are understood at this stage.

In the process we may have to answer certain questions such as:
1. What are the values, which should be provided to the program?
2. What is the order in which the input is to be provided?
3. What is the format of certain types of data (such as date)?
4. What is the accuracy of the input data?
5. What is the input device?
6. What is the range of values allowed for a particular input?

After the inputs we then analyse the outputs. For the outputs of a program we may have to answer certain question such as :
1. What are the outputs generated by the program?
(The outputs should be clearly and unambiguously specified).
2. What is the format of the outputs?
(Which includes type, accuracy and the units?)
3. What is the output device?
4. How the outputs should be displayed?
(Which includes spacing, layout and heading?)

DESIGN OF PROBLEM SOLUTIONS AND USE OF DESIGN TOOLS
Algorithms and **flowcharts** are two design tools, which help in the representation of a solution to a problem.
ALGORITHMS
An algorithm is a step-by-step procedure to solve a given problem.
The essential characteristics of an algorithm are
1. Every step of an algorithm should perform a single task.
2. Ambiguity (Confusion) should not be included at any stage in an algorithm.
3. An algorithm should involve a finite number of steps to arrive at a solution.
4. Simple statement and structures should be used in the development of the algorithm.
5. Every algorithm should lead to a unique solution of the problem.
6. An algorithm should have the capability to handle some unexpected situations, which may arise during the solution of a problem (For example, division by zero).

Example: *An algorithm to find the area of a triangle can be expressed as follows:*
Step 1 : Input height and base
Step 2 : Compute area = ½ *base *height
Step 3 : Display area

Algorithms may be developed for any type of problems; mathematical or scientific or business.
Normally, algorithms for mathematical involve mathematical formula, but algorithms for business problems are generally descriptive and have little use of formulas.

Steps Involved in developing an Algorithm
1. Clearly understand the problem statement so that a proper algorithm can be evolved.
2. Study the outputs to be generated so that the input can be specified.
3. Design the process, which will produce the desired result after taking the input.
4. Refine the process.
5. Test the algorithm by giving the test data and see if the desired output is generated. If not, make appropriate changes in the process and repeat the process.

Advantages of Algorithm
1. It is a step-by-step representation of a solution to a given problem, which is very easy to understand.
2. It has got a definite procedure, which can be executed within a set period of time.
3. It is easy to first develop an algorithm, and then convert it into a flowchart and then into a computer program.
4. It is independent of programming language.
5. It is easy to debug as every step has got its own logical sequence.

Disadvantages of Algorithm
It is time consuming and cumbersome as an algorithm is developed first which is converted into a flowchart and then into a computer program.

Example 1 : *Design an algorithm to calculate the Simple interest, given the principal (P), rate of interest (R) and time (T).*

```
Step 1 : Start
[ Input Principal, Time and Rate of interest ]
Step 2 : Input P, T and R
Step 3 : [ Compute Simple Interest ]
SI = P * T * R / 100
Step 4 : Display SI
Step 5 : Stop
```

Example 2 : *Design an algorithm to find the average of four numbers.*

```
Step 1 : Start
[ Input four numbers ]
Step 2 : Input A,B,C and D
Step 3 : [ Compute average of four numbers ]
AVERAGE = ( A + B + C + D ) / 4
Step 4 : Display AVERAGE
Step 5 : Stop
```

Example 3 : *Design an algorithm to find the largest of three numbers.*

```
Step 1 : Start
[ Input three numbers ]
Step 2 : Input A,B and C
[ Take first element as largest]
Step 3 : LARGE = A
Step 4 : If ( B > LARGE ) Then LARGE = B
Step 5 : If ( C > LARGE ) Then LARGE = C
Step 6 : Display LARGE
Step 7 : Stop
```

Example 4 : *Design an algorithm to print all numbers from 1 to N.*

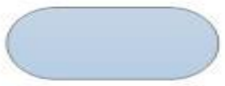
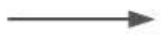
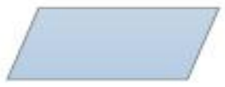
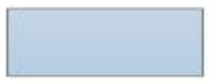
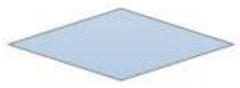
```
Step 1 : Start
[ Input upper limit of the series ]
Step 2 : Input N
[ Intitialize the first number ]
Step 3 : I =1
Step 4 : Display I
[ Increment counter by 1 ]
Step 5 : I = I + 1
Step 6 : If (I <= N) Then GOTO Step 4
Step 7 : Stop
```

FLOWCHARTS

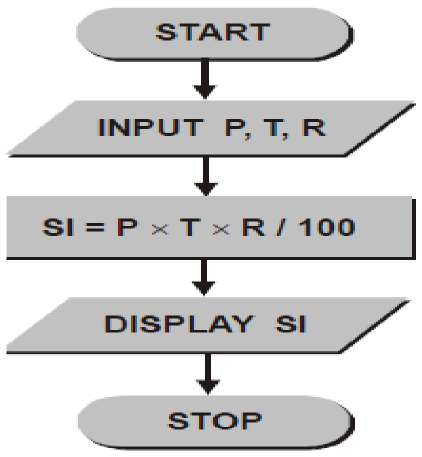
A flowchart is a pictorial or graphical representation of a solution to any problem. Flowcharts are constructed by using special geometrical symbols. Each symbol represents an activity.

The activity could be either input or output of data, computation or processing of data, taking a decision or terminating the solution, and so on. The symbols used in a flowchart are joined by arrows.

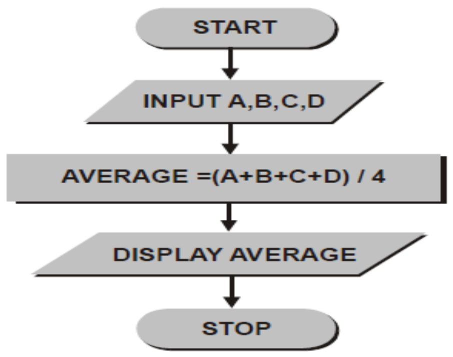
Symbols used in flowcharts

Symbol	Name	Function
	Start/end	An oval represents a start or end point
	Arrows	A line is a connector that shows relationships between the representative shapes
	Input/Output	A parallelogram represents input or output
	Process	A rectangle represents a process
	Decision	A diamond indicates a decision

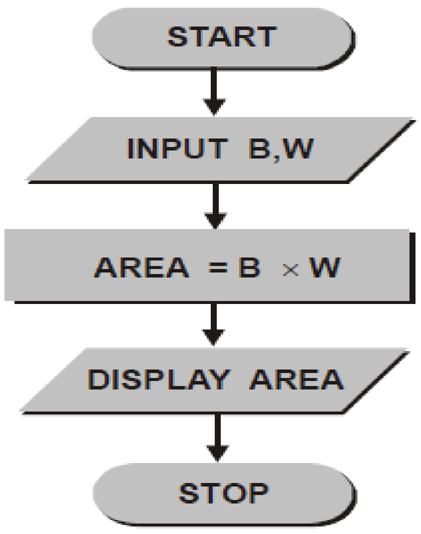
Example 1 : *Design a flowchart to calculate the Simple interest, given the principal (P), rate of interest (R) and time (T).*



Example 2: *Design a flowchart to find the average of four numbers.*

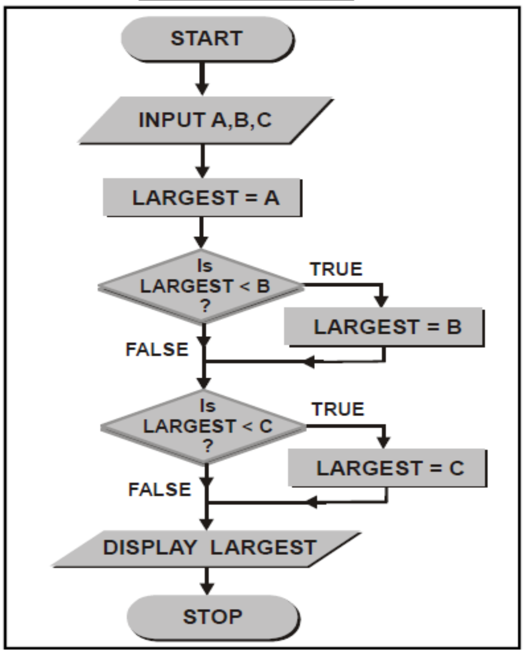


Example 3: *Design a flowchart to find the area of a rectangle. (Area = breadth width).*

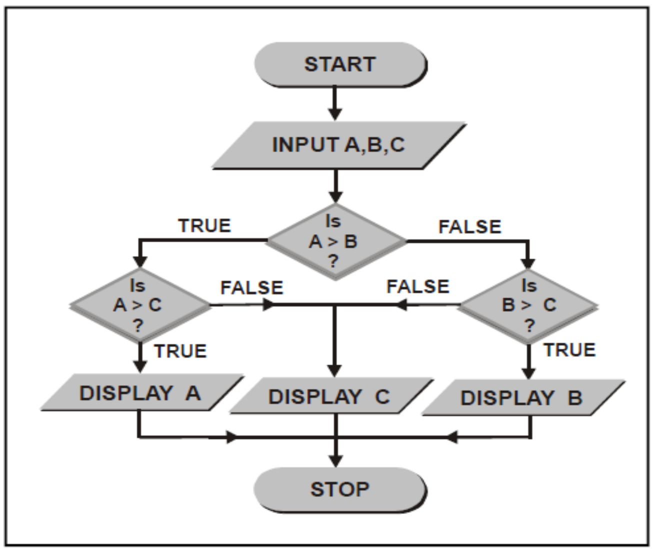


Example 4: *Design a flowchart to find the largest of three numbers.*

METHOD 1

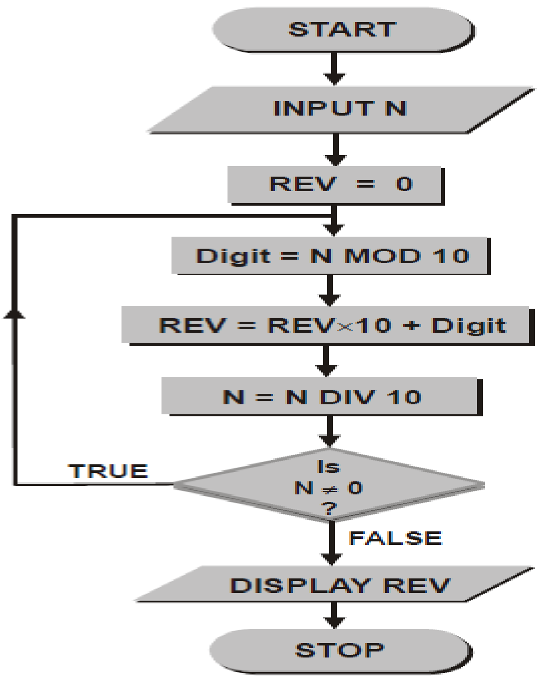


METHOD 2

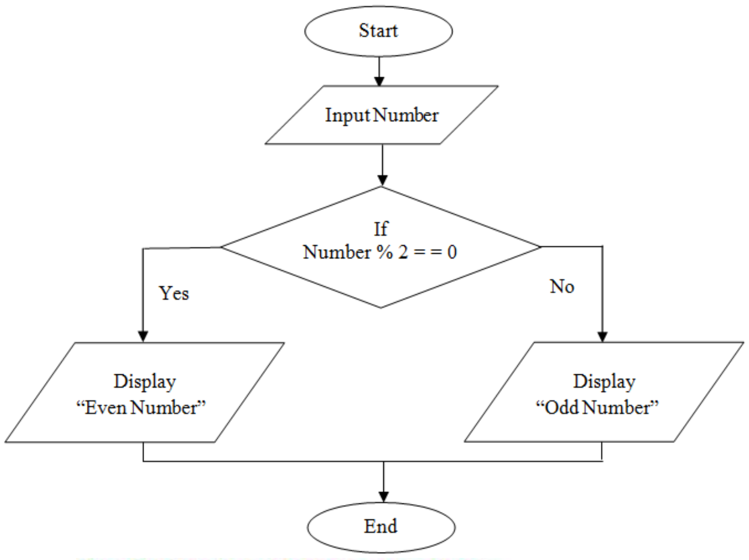


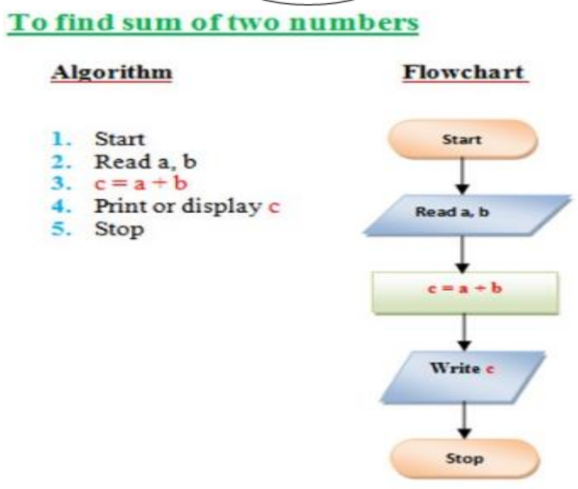
Example 5: *Develop an algorithm and a flowchart to reverse a given number.*

```
Step 1   :   Start
             [ Input a number ]
Step 2   :   Input N
Step 3   :   [ Initialize REV to 0 ]
             REV = 0
Step 4   :   Repeat Step 5 to Step 7 Until N = 0
Step 5   :   Digit = N MOD 10
Step 6   :   N = N DIV 10
Step 7   :   REV = REV×10 + Digit
             [End of Step 4 loop ]
Step 8   :   Display REV
Step 9   :   Stop
```



Example 6: *Develop a flowchart to find the odd or even number.*





To find sum of two numbers

Algorithm

1. Start
2. Read a, b
3. c = a + b
4. Print or display c
5. Stop

Flowchart

Steps Involved in developing Flowcharts

1. Clearly understand the problem statement so that a proper flowchart can be evolved.
2. Study the outputs to be generated so that the input can be specified.
3. Design the process, which will produce the desired result after taking the input.
4. Refine the process.
5. Test the flowchart by giving the test data and see if the desired output is generated. If not, make appropriate changes in the process and repeat the process.

Advantages of flowcharts

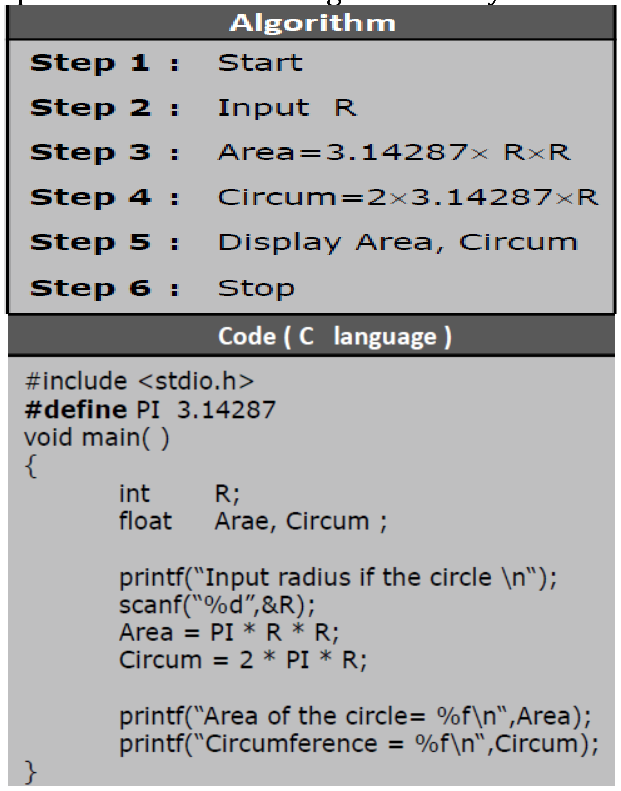
1. Flowcharts provide an excellent means of communication, which is very easy to understand.
2. It has got a definite procedure, which shows all the major parts of a program.
3. It is easy to convert it into a program.
4. It is independent of programming language.
5. It is easy to debug as every step has got its own logical sequence.

Disadvantages of flowcharts

1. It is time consuming and cumbersome as it requires the uses of a number of symbols which are to be properly represented.
2. The representation of a complex logic is very difficult in a flowchart.
3. Alterations and modifications can be only made by only redrawing the flowchart.

CODING

The process of converting the representation of a solution (either as an algorithm or flowchart) into a set of instructions in a programming language is referred to as coding.
The programmer takes algorithm or flowchart of each step and converts it into a proper representation using proper syntax and logic. The process is said to be successful or complete when the compiled version does not generate any error.



Algorithm	
Step 1 :	Start
Step 2 :	Input R
Step 3 :	Area=3.14287× R×R
Step 4 :	Circum=2×3.14287×R
Step 5 :	Display Area, Circum
Step 6 :	Stop

Code (C language)

```
#include <stdio.h>
#define PI 3.14287
void main( )
{
        int     R;
        float   Arae, Circum ;

        printf("Input radius if the circle \n");
        scanf("%d",&R);
        Area = PI * R * R;
        Circum = 2 * PI * R;

        printf("Area of the circle= %f\n",Area);
        printf("Circumference = %f\n",Circum);
}
```

DEBUGGING

The process of detecting and correcting errors in a program is referred to as **debugging**. Very few programs run correctly for the first time. Thus debugging is a very important and time-consuming phase of software development. The process of debugging ensures that the program does what the programmer intends to do. This stage is also referred to as **verification**.

The process of debugging requires us to follow the following sequence of steps:

1. The program is first compiled. During the process of compilation the compiler detects certain types of errors called as *syntax errors*.
2. All the detected errors are corrected and the program is recompiled. The process is repeated till no errors are displayed.
3. The program is then executed. During the execution of a program certain errors called as *semantic errors* are detected. Semantic errors normally occur due to the wrong use of logic.
4. Once all the errors are corrected the program is recompiled again and executed.

The following terms can be associated with the process of debugging.

Compiler	:	It is a software, which checks the entire program created by the user for a certain type of error called as syntax errors. If the program is error free the complete program is translated to its equivalent machine language program.
Source program	:	The program created by the user is referred to as source program.
Object program	:	The machine language program generated by the compiler is referred to as object program.
Syntax	:	It refers to the set of rules, which should be followed while creating every statement and structure in a program.
Semantic	:	It refers to the logic, which is followed while representing a solution.

TESTING

It is the process of checking whether the program works correctly according to the requirements of the user, i.e., whether the program generates the correct results for a given input of data.

The correctness of the program can be determined by trying a large number of carefully chosen data and then by seeing if the program generates the correct output in all those cases. If debugging is referred to as **verification**, testing is referred to as **validation**.

PROGRAM DOCUMENTATION

It is always said that a properly documented program can be easily reused again when needed, whereas an undocumented program requires so much effort that it is better to write a new program.

Def.: The written text and comments that make a program easier for others to understand, use, and modify. Or

Program documentation may be defined as the process of including comments or remarks at various stages in a program to improve the clarity and understanding of the program.

Comments are an integral part of program documentation. However there is no substitute for a good programming style. It is normally said that a program which is properly structured and has used properly chosen names to represent various elements is self-documented.

PROGRAM MAINTENANCE

The process of updating or providing new versions of a program so that the program meets the present day requirements of the user is often referred to as ***Program Maintenance***.
Program Maintenance may be required for the following reasons :
1. New errors or bugs, which were not detected during development and testing, were detected during the usage of the program.
2. The needs of the user have changed over a period of time and the program has to be modified to meet the present day needs.
3. The existing program is not functioning to the satisfaction of the user, thus the program has to be modified to provide the additional information.
4. The user has purchased a new computer or hardware and the program is not functioning properly on the new system.
5. The user has seen the use of new types of software using Graphical User Interface [**GUI**] and thus feels that his existing system has to be changed to the new system.

Program Development

Note: Write same steps as we discussed in problem solving

Modular design

Modular design, or "**modularity** in **design**", is an approach that subdivides a system into smaller parts called modules which can be independently created, modified, replaced or exchanged between different systems.
This independent work enables us to consistently improve our solutions and it leads to efficiency in development through reuse.