UNIT 2

Programming Methodology

UNIT 2

Chapter 1

Algorithms and Flowcharts

After studying this lesson, the students will be able to

- ✡ *understand the need of Algorithm and Flowcharts;*
- ✡ *solve problems by using algorithms and flowcharts;*
- ✡ *get clear idea about sequential, selection and iteration construct; and*
- ✡ *understand the finite- and infinite- loop.*

Introduction

Algorithm is a step-by-step process of solving a well-defined computational problem. In practice, in order to solve any complex real life problems, first we have to define the problem and then, design algorithm to solve it. Writing and executing a simple program may be easy; however, for executing a bigger one, each part of the program must be well organized. In short, algorithms are used to simplify the program implementation. The next step is making the flowchart. It is a type of diagram that represents an algorithm or process, showing the steps as 'boxes' of various kinds and their order by connecting them with arrows. Then, the flowchart will be converted into program code.

Algorithm

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. The instructions describe a computation, which will eventually produce output, when executed. We can use algorithm to solve any kind of problems. However, before writing a program, we need to write the steps to solve the problem in simple English language. This 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 magi 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 he is eligible to vote? (more than or equal to 18 years old).

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
```

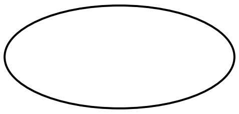
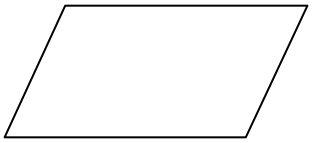
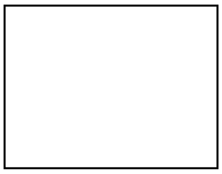
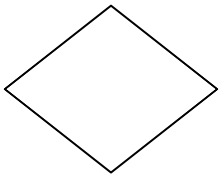
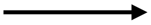
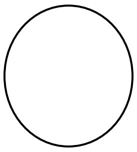
In the above example, steps 4, 5, 6 and 7 are executed more than one time.

Flowchart

In the previous section of this chapter, we have learnt to write algorithms, i.e. step-by-step process of solving a problem. We can also show these steps in graphical form by using some symbols. This is called flowcharting.

Flowchart Symbols

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

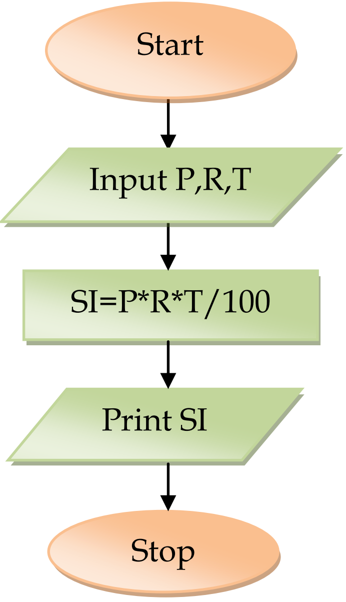
Symbols	Functions
1. 	Start/stop
2. 	Input/output
3. 	Processing
4. 	Decision Box
5. 	Flow of control
6. 	Connector

The following flowchart is an example of a sequential execution.

Example

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

Solution:

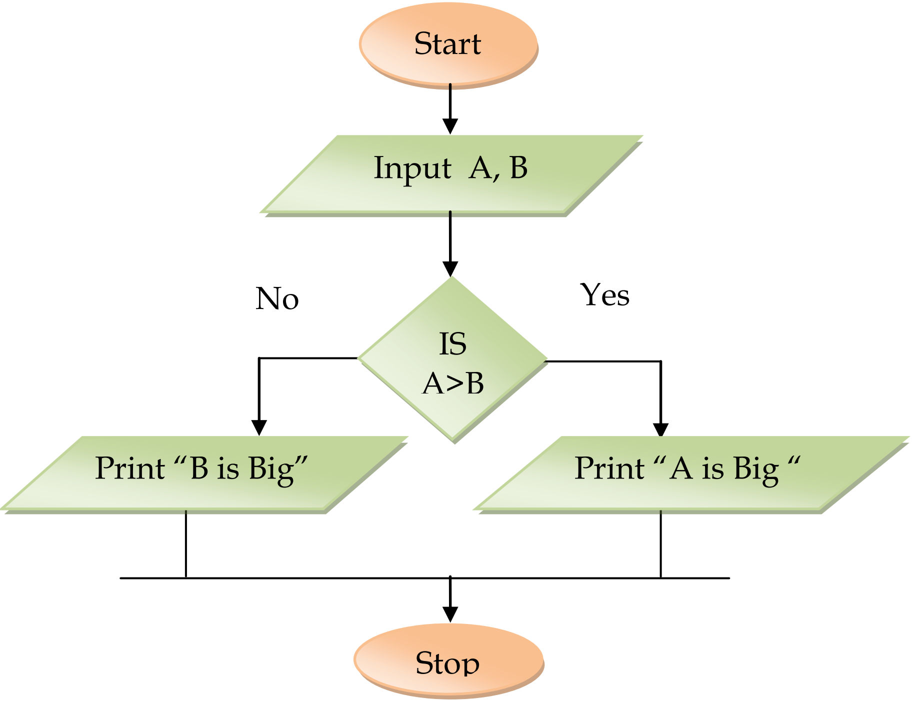


The following flowchart is an example of a selective execution.

Example

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

Solution:

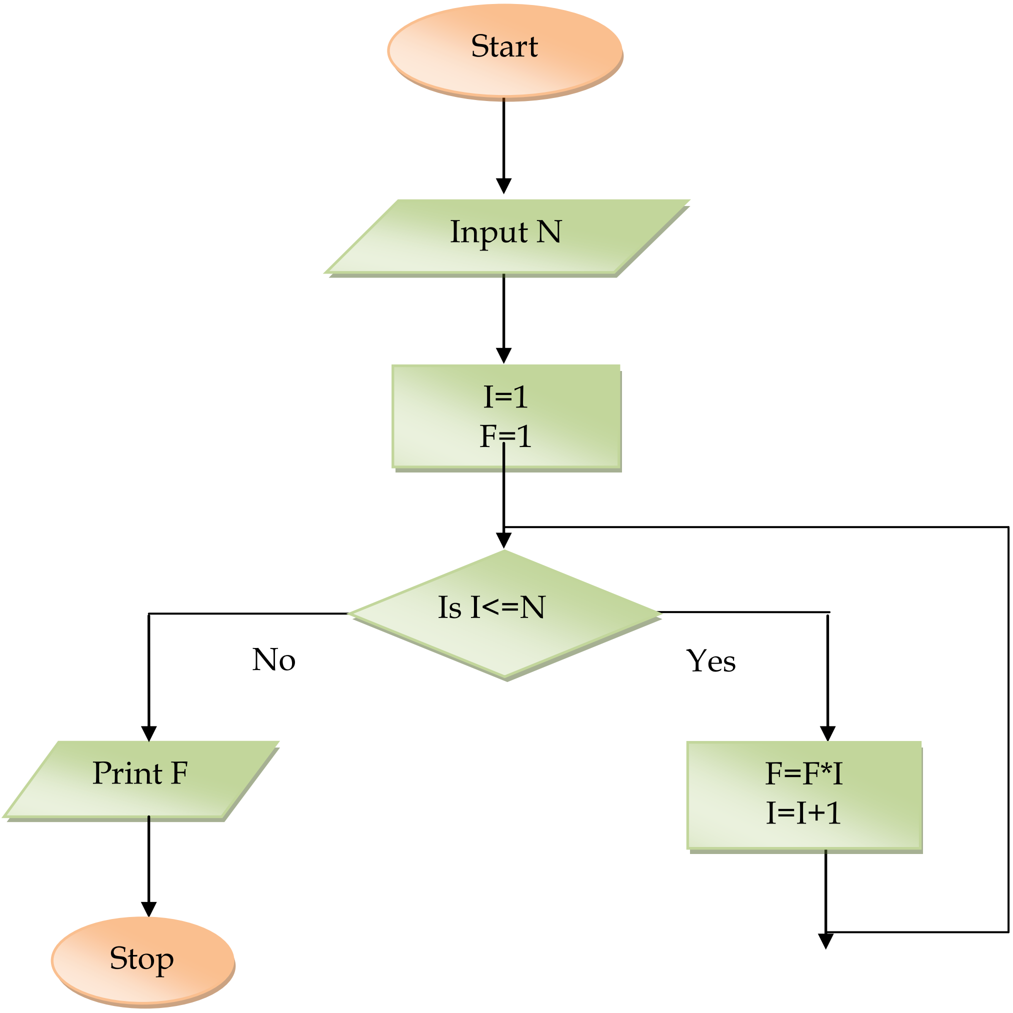


The following are the examples (VIII & IX) of an iterative execution.

Example

Draw a flow chart to find factorial of any number.

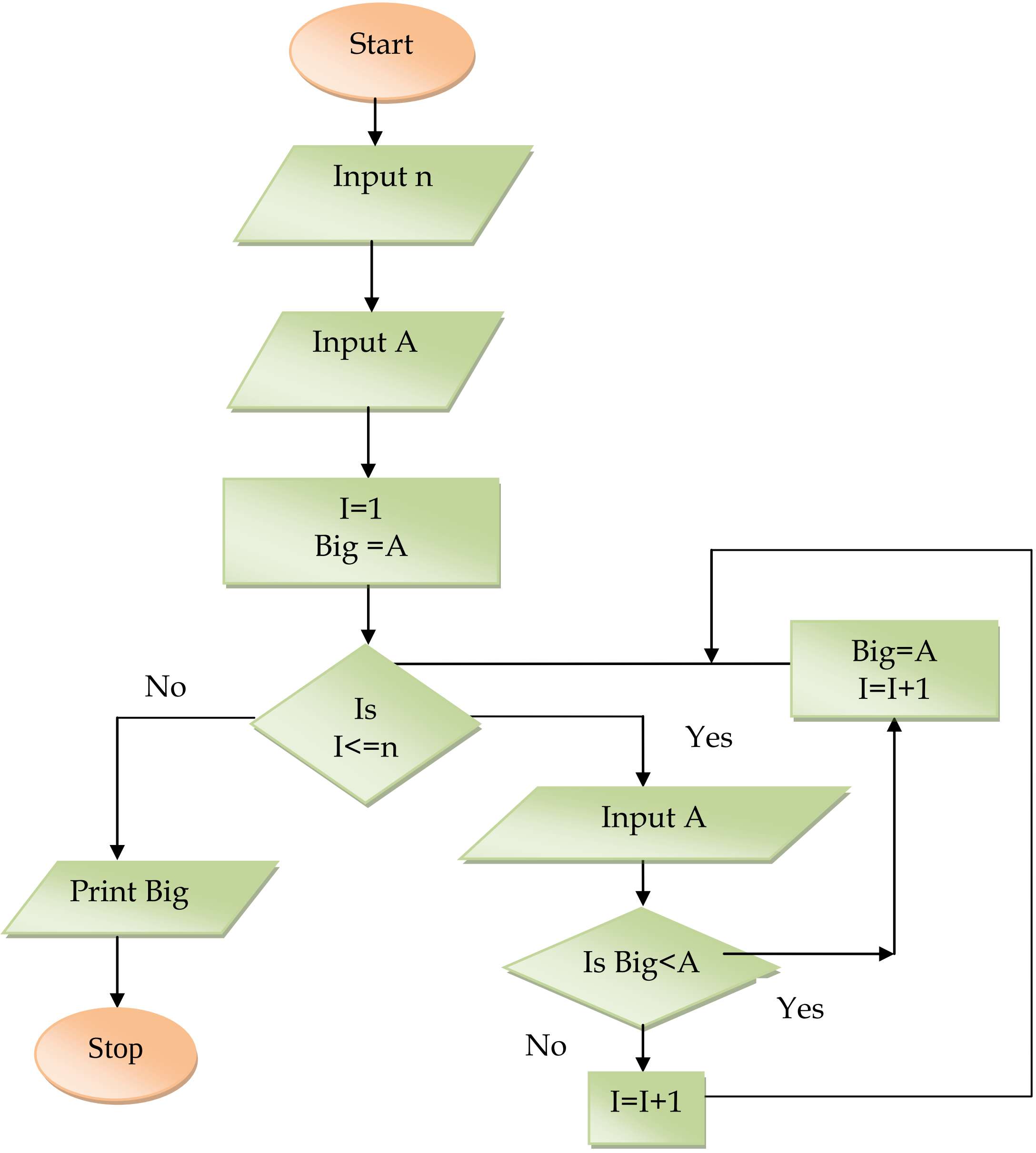
Solution:



Example

Draw a flow chart to find biggest number among ‘n’ numbers.

Solution:



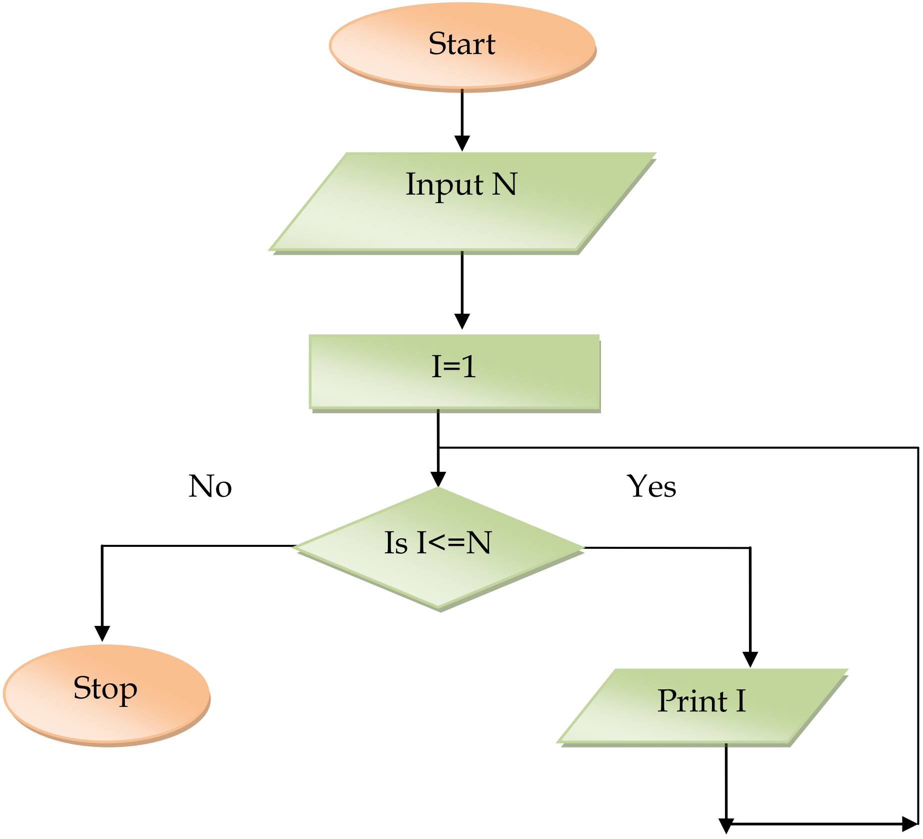
Finite and Infinite loop

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. The following is an example of infinite loop.

Example

Draw a flow chart to print the number from 1 to ∞.

Solution:



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

Note: Set of statements is executed again and again without any end is called infinite loop.

EXERCISE

Multiple choice questions:

1. A step by step method for solving a problem using English Language

   (a) program (b) Flowchart

   (c) statement (d) Algorithm

2. Set of statements is executed based upon conditional test.

   (a) Looping (b) Selective

   (c) Sequence (d) None

3. Set of statements is executed again and again based upon conditional test.

   (a) Looping (b) Selective

   (c) Sequence (d) None

4. The graphical representation of algorithm is

   (a) program (b) Flowchart

   (c) statement (d) Algorithm

5. All instructions are executed one after other.

   (a) Looping (b) Selective

   (c) Sequence (d) None

Answer the following questions.

1. Define Algorithm.
2. Define Flowchart.
3. Write an algorithm to find the sum of two numbers.
4. Write an algorithm to find the area of a triangle.

5. Write an algorithm to find whether given number is odd or even.

6. Write an algorithm to find the sum of all even number up to given number.

7. Draw a flowchart to find the area of a circle.

9. Draw a flowchart to find the smallest number among n numbers.

10. Draw a flowchart to find the sum of all multiples of 5 up to given number.

11. Mona is confused about finite loop and infinite loop, explain her with the help of example.

12. Write an algorithm and a flowchart to find sum of n numbers.

Chapter 2

Programming Methodology

After studying this lesson, the students will be able to

- *understand the need for good programs;*
- *understand how to solve problems using different ways;*
- *get clear idea about problem solving methodology; and*
- *understand the types of errors normally occur while writing programs.*

Introduction

Learning to write computer program is very much like learning any skill. First, we should understand the problems well and then try to solve it in a logical manner. For example: We have read many books available in the market for describing the car driving methods. However, we can learn driving once we actually get into the car and start driving it. The same logic is applied in computer programming also. Computer programming is the process of writing, testing, troubleshooting, debugging and maintaining of a computer program.

An effective program is that which gives result of all different inputs, including wrong input also. While creating program, we need to follow certain systematic approach. This systematic approach comprises two steps/things, viz., program structure and program representation. The program structure is implemented by using top-down or bottom-up approach and is known as 'popular approach', while the program representation plays an important role in making the program more readable and understandable.

What is a Good Program?

A Good Program means that it should produce correct and faster results, taking into account all the memory constraints. While making good program, we need to follow certain guidelines of programming language for creating a successful program. The following is the list of good programming habits that most people agree.

Clarity and Simplicity of Expression

Expressions are used to implement a particular task. It is a combination of Operators, Operands and Constants. Any expression used in the program should be understood by the user. The followings are some of the points to be kept in mind while using expressions in a program.

(i) Use library functions to make programs more powerful

Example

To find output = x^6

Output = X *X * X * X * X * X

We can use output = power (X, 6)

(ii) Follow simplicity to maintain the clarity of expression

Example

$$X = \frac{A+B}{A-B} - \frac{U+VY}{X+Y}$$

Then, we can write

X1 = (A+B) / (A-B)

X2 = (U+V*Y) / (X +Y)

X = X1 –X2

(iii) Avoid program tricks usage, whose meaning is difficult to understand by the user.

Use of proper names for identifiers

Identifiers are user defined names**.** They are used to name things. A name is associated with a function or data object (constants and variables) and used to refer to that function or data object. Identifiers are made up of letters (A-Z, a-z), digits (0-9), and the underscore character (_). They, however, must begin with a letter or underscore and not with a digit.

(i) Give meaningful name for variable (data – object) and function.

Example

To calculate Area of a Square

We use the variable names are Area and Side

Area = Side * Side.

(ii) Use proper names for constants.

Example

¶ = 3.14

Give Pi = 3.14

(iii) Do not use same name like custom, customer or account, accountant.

(iv) Do not use one letter identifiers.

Comments

A comment is a programming language construct, which is used to embed programmer-readable annotations in the source code of a computer program. Those annotations are potentially significant to programmers but typically ignorable to compilers and interpreters. Comments are usually added with the purpose of making the source code easy to understand. Hence, add comments to your code in simple English language that describes the function of the code and the reason for your decision to do it in a particular way as well. They are generally categorized as either ‘block comment’ or ‘line comment’. Block comment is implemented in python by “”” and “”” and line comment is implemented by #.

Example

"Write a program to print all numbers from 1 to 100 using while loop in python"

```
    A = 1
while (a<100):   # While statement
    print a
    a = a+1
```

Indentation

Leading white space (spaces and taps) at the beginning of each statement, which is used to determine the group of statement, is known as ‘indentation’.

Example

```
If  A > B :
        print  ‘A is  Big’        # Block1
else:
        print ‘B is Big’      # Block2
```

In the above example, **if statements** are a type of code block. If the ‘if’ expression evaluates to true, then Block1 is executed, otherwise, it executes Block2. Obviously, blocks can have multiple lines. As long as they are all indented with the same amount of spaces, they constitute one block.

Characteristics of good programming

Every computer needs proper instruction set (programs) to perform the required/assigned task. The quality of the program depends upon the instructions given to it. However, it is required to feed/provide the proper and correct instructions to the computer in order to yield/provide a correct and desired output. Hence, a program should be developed to ensure proper functionality of the computer and also should be easy to understand. A computer program should have some important characteristics, which are as follows:

Flexibility

A program should be flexible enough to handle most of the changes without having to rewrite the entire program. A flexible program is used to serve many purposes. For example, CAD (Computer Aided Design) software is used for different purposes such as; engineering drafting, printing circuit board layout and design, architectural design, technical drawing, industrial art, etc. Most of the programs are being developed for certain period and they need updation during the course of time.

User Friendly

A program that can be easily understood by a beginner is called 'user friendly'. It must interact with user through understandable messages. In addition, the proper message for the user to input data and to display the result, besides making the program easily understandable and modifiable.

Portability

Portability refers to the ability of an application to run on different platforms (operating systems) with or without minimal changes. Since the change of platform is a common phenomenon nowadays, due to the developments in hardware and the software, portability has to be taken care of it. In case, a program is developed for a particular platform, it would become obsolete after a certain period of time. At the same time, if a program that is developed does have the ability to work on different platforms, it makes software more useable. High language programs are often more portable than assembly language programs.

Reliability

It is the ability of a program to do its intended function accurately even if there are even small changes in the computer system. Moreover, the program must be able to handle unexpected situation like wrong input or no input. The programs, which save such ability are known as 'reliable'. For example, if the user does/gives wrong information to input, it should display a proper error message.

Self-Documenting Code

The source code, which uses suitable name for the identifiers (variables and methods), is called self-documenting code. Also, giving proper name for variables and methods would tell the reader of your code clearly -what is it doing? Hence, a good program must have a self-documenting code.

Problem solving process

The problem solving process starts with the problem specifications and ends with a concrete (and correct) program. Programming means a problem solving activity, which consists of four steps. They are;

(i) Understanding the problem;

(ii) Devising a plan;

(iii) Executing the plan; and

(iv) Evaluation

Understanding the problem

The first step is to understand the problem well. It may be very difficult to understand the problem but it is crucial. In general, one must find out the output from the given data (input data) and assess the relationship between input and output data. It is also important to verify whether the given information is sufficient to solve the problem or not.

Devising a plan

It means drawing an action plan to solve the problem, once understood. A plan is devised from data processing to the result according to the relationship that links both of them. If the problem is trivial, this step will not require much thinking.

Executing the plan

Once the plan is defined, it should follow the plan of action completely and each element of the plan should be checked as it is applied. In the course of execution, if any part of the plan is found to be unsatisfactory, the plan should be revised.

Evaluation

Finally, the result should be examined in order to make sure that it is valid and that the problem has been solved completely.

Problem solving methodology

As we all know, there are many methods/approaches available to solve a particular problem. However, the efficient way is to adopt a systematic method of problem solving. The use of systematic method of problem solving is crucial when we use a computer to solve a problem. We introduce here a seven steps problem solving method, which is closely related to the ***software life cycle*** (the various stages in the life of a program), that can be adapted by each person to solve the problem in their own style. They are given as under:

1. Problem Definition
2. Problem Analysis
3. Design the problem
4. Coding
5. Program Testing and Debugging
6. Documentation
7. Program Maintenance

Problem Definition/Specification (Theme)

Computer programs are written to solve problems posed by humankind. Prior to writing a program, one has to understand a description of the problem to solve. This description may be very precise or vague, but nevertheless, it is necessary/present. For instance, if you want to write a program to "Find the average of five numbers", you should ask yourself:

"What does average mean exactly?"

"How to calculate average value?"

Posing such questions compels you to define the problem very precisely. Once you are sure of what the problem entails, you must write down a list of specifications. *Specifications* are precise definitions of what the program must do. It must include the following at least:

- ✡ **Input:** what data must be included as input and in which form?
- ✡ **Output:** what data must the program produce and in which form? (in order to solve the problem)

Note: At the end of the problem definition step, you should have a list of specifications.

Problem Analysis

In this step, the problem has to be fragmented into smaller and manageable parts. The original problem has to be analyzed and divided into a number of sub-problems as these sub-problems are easier to solve and their solutions would become the components of the final program. Each sub-problem is divided into further smaller ones and this fragmentation has to be continued to achieve simple solutions. The use of modular programming is to get proper solution.

Modular Programming: Modular Programming is the act of designing and writing programs as functions (a large program is divided into the small individual components) that each one performs, a single well-defined function, which has minimal interaction between the sub-programs. It means that the content of each function is cohesive and there is low coupling between them. There are two methods available for modular programming. They are: top-down design and bottom-up design.

Top-Down design: The principles of top-down design dictate that a program should be divided into a main module and its related module. Each module should also be divided into sub modules according to software engineering and programming style. The division continues till the module consists only of an elementary process that is intrinsically understood and cannot be further sub-divided.

Bottom-up design: Bottom-up design is just the opposite of top-down design. It refers to a style of programming, in which, an application is constructed with existing primitives of the programming language and then gradually more and more complicated features are added till applications are written. In other words, initiating the design with simple modules and then build them into more complex structures ending at the top is bottom-up design.

Designing the problem

Designing the problem can be expressed in the form of

- Algorithm
- Flowchart

Algorithm: An algorithm is a set of instructions that describe a method for solving a problem. It is normally given in mix of computer code and English language. This is often called 'pseudo-code'.

Flowchart: The algorithm is represented in the form of a diagram with action boxes linked by lines showing the order in which they are executed. This is known as 'the flow of control'. It is the diagrammatic representation of an algorithm.

Coding

The process of translating the algorithm into syntax of a given language is known as 'Coding'. Since algorithm cannot be executed directly by the computer, it has to be translated into a *programming language.*

Program Testing and Debugging

Program Testing means running the program, executing all its instructions/ functions and testing the logic by entering sample data in order to check the output. Debugging is the process of finding and correcting the errors in the program code.

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

```
a = 0
while a < 10
    a = a + 1
    print a
```

In the above statement, the second line is not correct. Since the while 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.

```
a = 100
while a < 10:
    a = a + 1
    print 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 running 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. This type of error is called runtime error.

Example

(a)

```
A=10
B=0
print A/B
```

(b) During running time, if we try to open a file that does not exist in the hard disk, then it will create runtime error.

Documentation

The documentation includes the problem definition, design documents, a description of the test perform, a history of the program development and its different versions and a user’s manual. Such a manual is designed for a naive user and illustrates the preparation of input data, running the program and obtaining & interpreting the results.

Program maintenance

It is not directly part of the original implementation process, but needs special emphasis. All activities that occur after a program operation are part of the program maintenance. Many large programs have long life span that often exceed the lifetime of the hardware they run on. Usually, the expenditure for the program maintenance will be more than the developmental cost of the program. The program maintenance includes the following:

- Finding and eliminating previously undetected program errors;
- Modifying the current program, often to improve its performance, or to adapt to new laws or government regulations, or to adapt to a new hardware, or to a new operating system;
- Adding new features or a better user interface, or new capabilities to the program; and
- Updating the documentation.

Maintenance is an important part of the life cycle of a program. It is also important as far as documentation is concerned, since any change pertaining to a program will require updating of internal as well as external documentation. Maintenance documentation will include results of the program development steps, design documents, program code and test information.

EXERCISE

Multiple choice questions:

1. User Define name.

   (a) Identifier (b) constant

   (c) syntax (d) expression

2. If we overcome the rules of the programming language, we get

   (a) Runtime error (b) Syntax error

   (c) logical error (d) None of the above.

3. Correcting the program code:

   (a) Testing (b) Syntax error

   (c) Runtime error (d) Debugging

4. Designing the problem

   (a) Testing (b) Debugging

   (c) logical error (d) Algorithm

5. Algorithm when translated into a programming language is called

   (a) Flowchart (b) Identifier

   (c) Code (d) Debugging

6. The program must be able to handle unexpected situation like wrong input or no input.

   (a) Error (b) Expression

   (c) Portability (d) Reliability

7. Leading white space at the beginning of each statement, which is used to determine the group of statement.

   (a) Testing (b) Indentation

   (c) Debugging (d) None of the above

8. It refers to the ability of an application to run on different platforms with or without minimal changes.

   (a) Error (b) Flexibility

   (c) Portability (d) Reliability

9. It is a combination of Operators, Operands and Constants.

   (a) Identifier (b) Expression

   (c) Syntax (d) Task

10. Each module should also be divided into sub modules according to software engineering and programming style.

   (a) Top down method (b) Bottom up method

   (c) Coding (d) None of the above

Answer the following questions.

1. What is a good program?
2. What is an identifier?
3. How to write comments in a program?
4. What is the purpose of expression? Explain with an example.
5. Write and explain all steps of programming methodology.
6. Differentiate between runtime errors and logical errors.
7. Define documentation.
8. What is program maintenance?
9. Define modular programming.
10. Differentiate between top down and bottom up methods of modular programming.
11. Explain types of errors with examples.
12. How to maintain programs?

13. Write all steps of program methodology?

14. What do you mean by clarity and simplicity of expression?

15. What do you mean by flexibility?

16. Explain all steps of problem solving process.

17. What is indentation? Explain with an example.

18. What do you mean by debugging?

19. What is the use of self documenting code in programming?

20. What is the purpose of giving meaningful name for identifiers?