3

Principles of Programming and Problem Solving

Key concepts

- **Problem solving using computers**
- **Approaches in problem solving**
  - Top down design
  - Bottom up design
- **Phases in programming**
  - Problem identification
  - Algorithms and Flowcharts
  - Coding the program
  - Translation
  - Debugging
  - Execution and testing
  - Documentation
- **Performance evaluation of algorithms**

We have learnt the concept of data processing and the role of computers in data processing. We have also discussed the computer as a system with components such as hardware, software and users. We had a detailed discussion on these components in the previous chapter. Let us recall the definition of software. In its simplest form, we can say that software means a collection of programs to solve problems using computers. As we know, a computer cannot do anything on its own. It must be instructed to perform the desired job. Hence it is necessary to specify a sequence of instructions that the computer must perform to solve a problem. Such a sequence of instructions written in a language that is understood by a computer is called a "computer program". Writing a computer program is a challenging task. However, we can attempt it by procuring the concepts of problem solving techniques and different stages of programming.

3.1 Problem solving using computers

A computer can solve problems only when we give instructions to it. If it understands the tasks contained in the instructions, it will work accordingly. An instruction is an action oriented

statement. It tells the computer what operation it should perform. A computer can execute (carry out the task contained in) an instruction only if the task is specified precisely and accurately. As we learnt in the previous chapter, there are programmers who develop computer programs for solving problems. Once the program is developed and stored permanently in a computer, we can ask the computer to execute it as and when required.

We should be cautious about the clarity of the logic of the solution and the format of instructions while designing a program, because computer does not possess common sense or intuition. As human beings we use judgments based on experience, often on subjective and emotional considerations. Such value oriented judgments often depend on what is called "common sense". As opposed to this, a computer exhibits no emotion and has no common sense. That is why we say that computer has no intelligence of its own.

In a way, computer may be viewed as an 'obedient servant'. Being obedient without exercising 'common sense' can be very annoying and unproductive. Take the instance of a master who sent his obedient servant to a post office with the instruction "Go to the post office and buy ten 5 rupees stamps". The servant goes to the post office with the money and does not return even after a long time. The master gets worried and goes in search of him to the post office and found the servant standing there with the stamps in his hand. When the angry master asks the servant for an explanation, the servant replies that he was ordered to buy ten 5 rupees stamps but not to return with them!

3.2 Approaches in problem solving

A problem may be solved in different ways. Even the approach may be different. In our life, we may seek medical treatment for some diseases. We can consult an allopathic, ayurvedic or homoepathic doctor. Each of their approaches may be different, though all of them are solving the same problem. Similarly in problem solving also different approaches are followed. Let us discuss the two popular designing styles of problem solving – Top down design and Bottom up design.

3.2.1 Top down design

Look at Figure 3.1. If you are asked to draw this picture, how will you proceed? It may be as follows:

1. Draw the outline of the house.
2. Draw the chimney
3. Draw the door
4. Draw the windows

Fig. 3.1 : Lay-out of a House

While drawing the door in Step 3, the procedure may be as follows:

3.1 Outline of the door

3.2 Shading

3.3 Handle

Similarly the windows may be drawn as follows:

4.1 Outline of the window

4.2 Shading

4.3 Horizontal and Vertical lines

The procedure described above may be summarised as follows:

The whole problem (here drawing the picture) is broken down into smaller tasks. Thus four tasks are identified to solve the problem. Some of these tasks (here steps 3 and 4 for drawing the door and windows) are further subdivided. Thus any complex problem can be solved by breaking it down into different tasks and solving each task by performing simpler activities. This concept is known as **top down design** in problem solving.

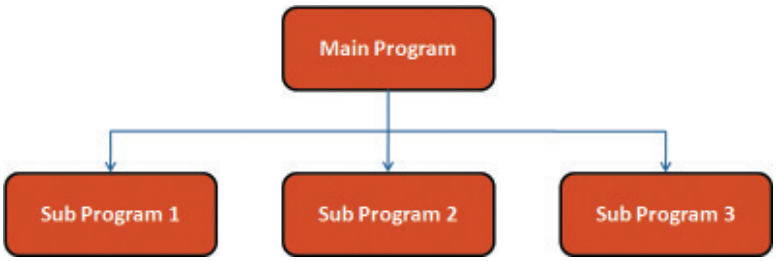


Fig. 3.2 : Decomposition of a problem

It is one of the programming approaches that has been proven the most productive. As shown in Figure 3.2, top down design is the process of breaking the overall procedure or task into component parts (modules) and then subdividing each component module until the lowest level of detail is reached. It is also called top down decomposition since we start "at the top" with a general problem and design specific solutions to its sub problems. In order to obtain an effective solution for the main problem, it is desirable that the sub problems (sub programs) should be independent of each other. So, each sub problem can be solved and tested independently.

The following are the advantages of problem solving by decomposition:

- Breaking the problem into parts helps us to clarify what is to be done in each part.
- At each step of refinement, the new parts become less complicated and therefore, easier to figure out.
- Parts of the solution may turn out to be reusable.
- Breaking the problem into parts allows more than one person to work for the solution.

3.2.2 Bottom up design

Consider the case of constructing a house. We do not follow the top down design, but the bottom up design. The foundation will be the first task and roofing will be the last task. Breaking down of tasks is carried out here too. It is true that some tasks can be carried out only after the completion of some other tasks. However roofing which is the main task will be carried out only after the completion of bottom level tasks.

Similarly in programming, once the overall procedure or task is broken down into component parts (modules) and each component module is further sub divided until the lowest level of detail has been reached, we start solving from the lowest module onwards. The solution for the main module will be developed only after designing specific solutions to its sub modules. This style of approach is known as **bottom up design** for problem solving. Here again, it is desirable that the sub problems (subprograms) should be independent of each other.

Let us do *Youth festivals are conducted every year in our schools. Usually the duties and responsibilities are decomposed. Discuss how the tasks are divided and executed to organise the youth festival successfully.*

3.3 Phases in programming

As we have seen, problem solving using computer is a challenging task. A systematic approach is essential for this. The programs required can be developed only by going through different stages. Though we have in-born problem solving skills, it can be applied effectively only by proper thinking, planning and developing the logical reasoning to solve the problem. We can achieve this by proceeding through the following stages, in succession:

1. Problem identification
2. Preparing algorithms and flow-charts

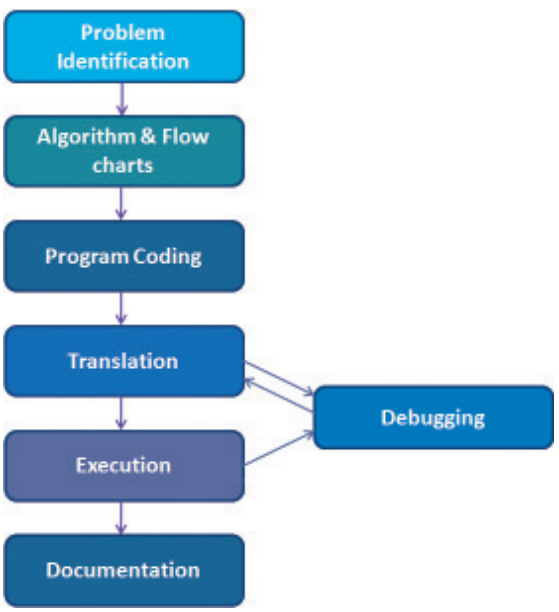


Fig. 3.3 : Phases of Programming

3. Coding the program using programming language
4. Translation
5. Debugging
6. Execution and testing
7. Documentation

Figure 3.3 shows the order of performing the tasks in various stages of programming. Note that the debugging stage is associated with both translation and execution. The activities involved in the seven stages mentioned above are detailed in the following sections.

3.3.1 Problem identification

Let us discuss a real life situation; suppose you are suffering from a stomach ache. As you know this problem can be solved by a doctor. Your doctor may ask you some questions regarding the duration of pain, previous occurrence, your diet etc., and examine some parts of your body using the stethoscope, X-ray or scan. All these are a part of the problem study. After these procedures, your doctor may be able to identify the problem and state it using some medical term. Now the second stage begins with the derivation of some steps for solution known as prescriptions.

It is clear that before deriving the steps for solution, the problem must be analysed. During this phase you will be able to identify the data involved in processing, its type and quantity, formula to be used, activities involved, and the output to be obtained. Once we studied the problem clearly, and are convinced about the sequence of tasks required for the solution, we can go to the next phase. This is the challenging phase as it exploits the efficiency of the programmer (problem solver).

3.3.2 Algorithms and Flowcharts

Once the problem is identified, it is necessary to develop a precise step-by-step procedure to solve the problem. This procedure is not new or confined to computers. It has been in use for a very long time, and in almost all walks of life. One such procedure taken from real life is described below. It is a cooking recipe of an omlette taken from a magazine.

Ingredients

Eggs - 2 Nos, Onion - 1 (small sized, chopped); Green chili - 2 (finely chopped); Oil - 2 tea spoon, Salt - a pinch.

Method

Step 1 : Break the eggs and pour the contents in a vessel and stir.

Step 2 : Mix chopped onion, green chilies and salt with the stirred egg.

Step 3 : Place a pan on the stove and light the stove.

Step 4 : Pour the oil in the pan and wait till it gets heated.

Step 5 : Pour the mixture prepared in step 2 into the pan and wait till the side is fried.

Step 6 : Turn over to get the other side fried well.

Step 7 : Take it out after some seconds.

Result

An omlette is ready to be served with pepper powder.

The recipe given above has the following properties:

1. It begins with a list of ingredients required for making the omlette. These may be called the inputs.
2. A sequence of instructions is given to process the inputs.
3. As a result of carrying out the instructions, some outputs (here, omlette) are obtained.

The instructions given to process the inputs are, however, not precise. They are ambiguous. For example, the interpretation of "till the side is fried" in step 5 and "fried well" in step 6 can vary from person to person. Due to such imprecise instructions, different persons following the same recipe with the same inputs can produce omlettes which differ in size, shape and taste.

The above ambiguities should be avoided while writing steps to solve the problems using the computer.

a. Algorithm

Mathematicians trace the origin of the word algorithm to the name of a famous Arab mathematician, Abu Jafar Mohammed Ibn Musaa Al-Khowarizmi. The word 'algorithm' is derived from the last part of his name Al-Khowarizmi. In computer terminology an **algorithm** may be defined as a finite sequence of instructions to solve a problem. It is a step-by-step procedure to solve a

Fig.3.4 : Abu Jafar Mohammed Ibn Musaa Al-Khowarizmi (780 - 850)

problem, where each step represents a specific task to be carried out. However, in order to qualify as an algorithm, a sequence of instructions must possess the following characteristics:

(i) It should begin with instruction(s) to accept inputs. These inputs are processed by subsequent instructions. Sometimes, the data to be processed will be given along with the problem. In such situations, there will be no input instruction at the beginning of the algorithm.

(ii) Use variables to refer the data, where variables are user-defined words consisting of alphabets and numerals that are similar to those used in mathematics. Variables must be used for inputting data and assigning values or results.

(iii) Each and every instruction should be precise and unambiguous. In other words, the instructions must not be vague. It must also be possible to carry them out. For example, the instruction "Catch the day" is precise, but cannot be carried out.

(iv) Each instruction must be sufficiently basic such that it can, in principle, be carried out in finite time by a person with paper and pencil.

(v) The total time to carry out all the steps in the algorithm must be finite. As algorithm may contain instructions to repetitively carry out a group of instructions, this requirement implies that the number of repetitions must be finite.

(vi) After performing the instructions given in the algorithm, the desired results (outputs) must be obtained.

To gain insight into algorithms, let us consider a simple example. We have to find the sum and average of any three given numbers. Let us write the procedure for solving this problem. It is given below:

Step 1: Input three numbers.
Step 2: Add these numbers to get the sum
Step 3: Divide the sum by 3 to get the average
Step 4: Print sum and average

Though the procedure is correct, while preparing an algorithm, we have to follow any of the standard formats. Let us see how the procedure listed above can be written in an algorithm style.

Example 3.1: Algorithm to find the sum and average of three numbers

Let A, B, C be variables for the input numbers; and S, Avg be variables for sum and average.

Step 1: Start

Step 2: Input A, B, C

Step 3: S = A + B + C

Step 4: Avg = S / 3

Step 5: Print S, Avg

Step 6: Stop

The above set of instructions qualifies as an algorithm for the following reasons:

- It has input (The variables A, B and C are used to hold the input data).
- The processing steps are precisely specified (Using proper operators in Steps 3 and 4) and can be carried out by a person using pen and paper.
- Each instruction is basic (Input, Print, Add, Divide) and meaningful.
- It produces two outputs such as sum (S) and average (Avg).
- The beginning and termination points are specified using Start and Stop.

Types of instructions

As we know, a computer can perform only limited types of operations. So we can use only that many instructions to solve problems. Before developing more algorithms, let us identify the types of instructions constituting the algorithm.

- Computer can accept data that we input. So, we can use input instructions. The words **Input**, **Accept** or **Read** may be used for this purpose.
- Computer gives the results as output. So we can use output instructions. The words **Print**, **Display** or **Write** may be used for this purpose.
- Data can directly be stored in a memory location or data may be copied from one location to another. Similarly, results of arithmetic operations on data can also be stored in memory locations. We use assignment (or storing) instruction for this, similar to that used in mathematics. Variables followed by the equal symbol (=) can be used for storing value, where variables refer to memory locations.
- A computer can compare data values (known as logical operation) and make decisions based on the result. Usually, the decision will be in the form of selecting or skipping a set of one or more statements or executing a set of instructions repeatedly.

b. Flowcharts

An idea expressed in picture or diagram is preferred to text form by people. In certain situations, algorithm may be difficult to follow, as it may contain complex tasks and repetitions of steps. Hence, it will be better if it could be expressed in pictorial form. The pictorial representation of an algorithm with specific symbols for instructions and arrows showing the sequence of operations is known as **flowchart**. It is primarily used as an aid in formulating and understanding algorithms. Flowcharts commonly use some basic geometric shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and concise statements. These boxes are connected by solid lines with arrow marks to indicate the flow of operation; that is, the exact sequence in which the instructions are to be executed.

Normally, an algorithm is converted into a flowchart and then the instructions are expressed in some programming language. The main advantage of this two-step approach in program writing is that while drawing a flowchart one is not concerned with the details of the elements of programming language. Hence he/she can fully concentrate on the logic (step-by-step method) of the procedure. Moreover, since a flowchart shows the flow of operations in pictorial form, any error in the logic of the procedure can be detected more easily than in a program. The algorithm and flow chart are always a reference to the programmer. Once these are ready and found correct as far as the logic of the solution is concerned, the programmer can concentrate on coding the operations following the constructs of programming language. This will normally ensure an error-free program.

Flowchart symbols

The communication of program logic through flowcharts is made easier through the use of symbols that have standardised meanings. We will only discuss a few symbols that are needed to indicate the necessary operations. These symbols are standardized by the American National Standards Institute (ANSI).

1. Terminal

As the name implies, it is used to indicate the beginning (START) and ending (STOP) in the program logic flow. It is the first symbol and the last symbol in the flowchart.

It has the shape of an ellipse. When it is used as START, an exit flow will be attached. But when it is used as a STOP symbol, an entry flow will be attached.

2. Input / Output

The parallelogram is used as the input/output symbol. It denotes the function of an input/output device in the program. All the input/output instructions are expressed using this symbol. It will be attached with one entry flow and one exit flow.

3. Process

A rectangle is used to represent the processing step. Arithmetic operations such as addition, subtraction, multiplication, division as well as assigning a value to a variable are expressed using this symbol. Assigning a value to a variable means moving data from one memory location to another (e.g. a=b) or moving the result from ALU to memory location (e.g. a=b+5) or even storing a value to a memory location (e.g. a=2). Process symbol also has one entry flow and one exit flow.

4. Decision

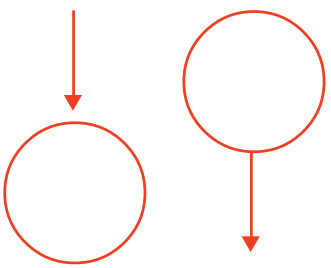
The rhombus is used as decision symbol and is used to indicate a point at which a decision has to be made. A branch to one of two or more alternative points is possible here. All the possible exit paths will be specified, but only one of these will be selected based on the result of the decision. Usually this symbol has one entry flow and two exit flows - one towards the action based on the truth of the condition and the other towards the alternative action.

5. Flow lines

Flow lines with arrow heads are used to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed. The normal flow is from top to bottom and left to right. But in some cases, it can be from right to left and bottom to top. Good practice also suggests that flow lines should not cross each other and such intersections should be avoided wherever possible.

6. Connector

When a flowchart becomes bigger, the flow lines start criss-crossing at many places causing confusion and reducing comprehension of the flowchart. Moreover, when the flowchart becomes too long to fit into a single page the use of flow lines becomes impossible. Whenever a flowchart



becomes complex and the number and direction of flow lines is confusing or it spreads over more than one page, a pair of connector symbols can be used to join the flow lines that are broken. This symbol represents an "entry from", or an "exit to" another part of the flowchart. A connector symbol is represented by a circle and a letter or digit is placed within the circle to indicate the link. A pair of identically labelled connector symbols is commonly used to indicate a continuous flow. So two connectors with identical labels serve the same function as a long flow line. In a pair of identically labelled connectors, one shows an exit to some other chart section and the other indicates an entry from another part of the chart.

Figure 3.5 shows that flowchart of the problem discussed in Example 3.1.

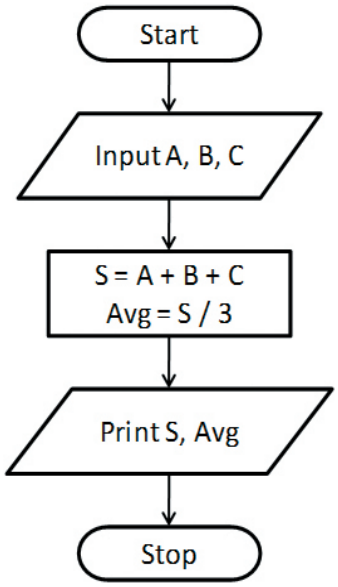


Fig. 3.5 : Flow chart for Sum and Average

The instruction given in each step in the algorithm is represented using the concerned symbol. Each symbol is labelled properly with the respective instruction. The flow of operations is clearly shown using the flow lines.

Advantages of flowcharts

Flowcharts are beneficial in many ways in program planning.

- **Better communication:** Since a flowchart is a pictorial representation of a program, it is easier for a programmer to explain the logic of the program to some other programmer through a flowchart rather than the program itself.
- **Effective analysis:** The whole program can be analysed effectively through the flowchart as it clearly specifies the flow of the steps constituting the program.
- **Effective synthesis:** If a problem is divided into different modules and the solution for each module is represented in flowcharts separately, they can finally be placed together to visualize the overall system design.
- **Efficient coding:** Once a flowchart is ready, programmers find it very easy to write the concerned program because the flowchart acts as a road map for them. It guides them to go from the starting point of the program to the final point ensuring that no steps are omitted.

Limitations of flowcharts

In spite of their many obvious advantages, flowcharts have some limitations:

- Flowcharts are very time consuming and laborious to draw with proper symbols and spacing, especially for large complex algorithms.
- Owing to the symbol-string nature of flowcharting, any change or modification in the logic of the algorithm usually requires a completely new flowchart.
- There are no standards determining the amount of detail that should be included in a flowchart.

Now let us develop algorithms and draw flowcharts for solving various problems.

Example 3.2: To find the area and perimeter of a rectangle

We know that this problem can be solved, if the length and breadth of the rectangle are given. The result can be obtained by using the following formula:

Perimeter = 2 (Length + Breadth), Area = Length × Breadth

Let L and B be variables for length and breadth; and P, A be variables for perimeter and area.

Step 1: Start
Step 2: Input L, B
Step 3: P = 2 * (L + B)
Step 4: A = L * B
Step 5: Print P, A
Step 6: Stop

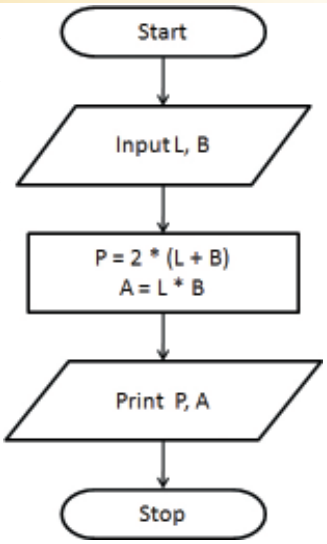


Fig. 3.6 : Flow chart for Area and Perimeter

The flowchart is given in Figure 3.6.

The algorithms developed in Examples 3.1 and 3.2 consist of six instructions each. In both the cases, the instructions will be executed one by one in a sequential fashion as shown in Figure 3.7. The order of execution of instructions is known as flow of control. We can say that the two algorithms follow in a sequential flow of control.

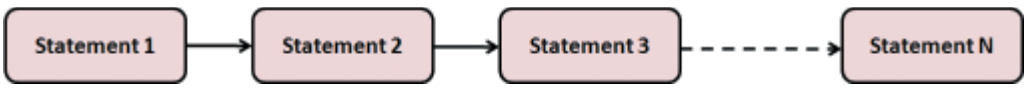


Fig. 3.7 : Sequential flow of control

Let us do

Develop an algorithm and draw the flow chart to input a time in seconds and convert it into the Hr: Min: Sec format. (For example, if 3700 is given as input, the output should be 1 Hr: 1 Min: 40 Sec).

Example 3.3: Find the height of the taller one among two students

Here, two numbers representing the height of two students are to be input. The larger number is to be identified as the result. We know that a comparison between these numbers is to be made for this. The algorithm is given below:

Step 1: Start
Step 2: Input H1, H2
Step 3: If H1 > H2 Then
Step 4: Print H1
Step 5: Else
Step 6: Print H2
Step 7: End of If
Step 8: Stop

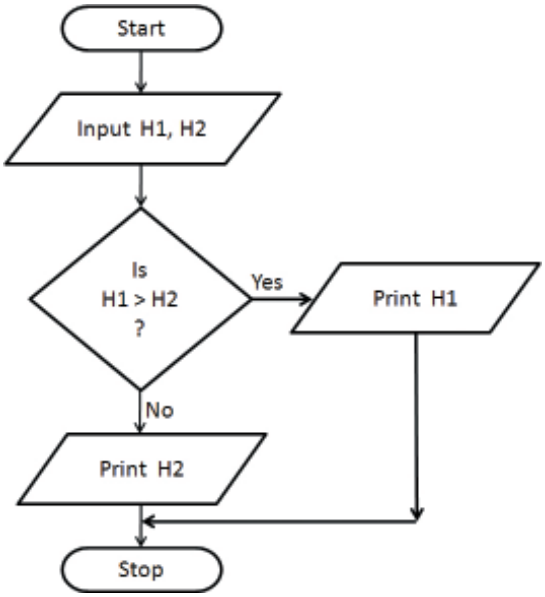


Fig. 3.8 : Flowchart to find larger value

The flowchart of this algorithm is shown in Figure 3.8. This algorithm uses the decision making aspect. In step 3, a condition is checked. Obviously the result may be True or False based on the values of variables H1 and H2. The decision is made on the basis of this result. If the result is True, step 4 will be selected for execution, otherwise step 6 will be executed. Here one of the two statements (either step 4 or step 6) is selected for execution based on the condition. A branching is done in step 3. That is, this algorithm uses the selection structure to solve the problem. As shown in Figure 3.9, the condition branches the flow to one of the two sets based on the result of condition.

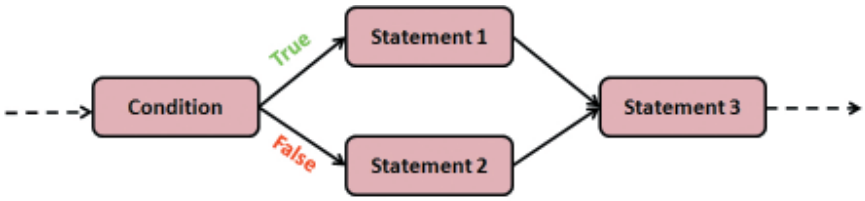


Fig. 3.9 : Selection structure

The working of selection construct is as shown in Figure 3.10. The flow of control comes to the condition; it will be evaluated to True or False. If the condition is True, the instructions given in the true block will be executed and false block will be skipped. But if the condition is False, the true block will be skipped and the false block will be executed. Now let us solve another problem.

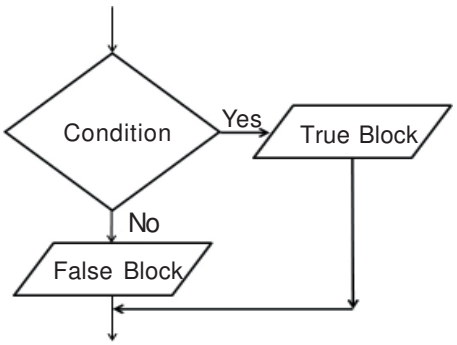


Fig. 3.10 : Flowchart of selection

Example 3.4: To input the scores obtained in 3 unit-tests and find the highest score

Here we have to input three numbers representing the scores and find the largest number among them. The algorithm is given below and flowchart is shown in Figure 3.11.

Step 1: Start

Step 2: Input M1, M2, M3

Step 3: If M1>M2 And M1>M3 Then

Step 4: Print M1

Step 5: Else If M2 > M3 Then

Step 6: Print M2

Step 7: Else

Step 8: Print M3

Step 9: End of If

Step 10: Stop

This algorithm uses multiple branching based on different conditions. Three different actions are provided, but only one of them is executed. Another point we have to notice is that the first condition consists of two comparisons. This kind of condition is known as compond condition.

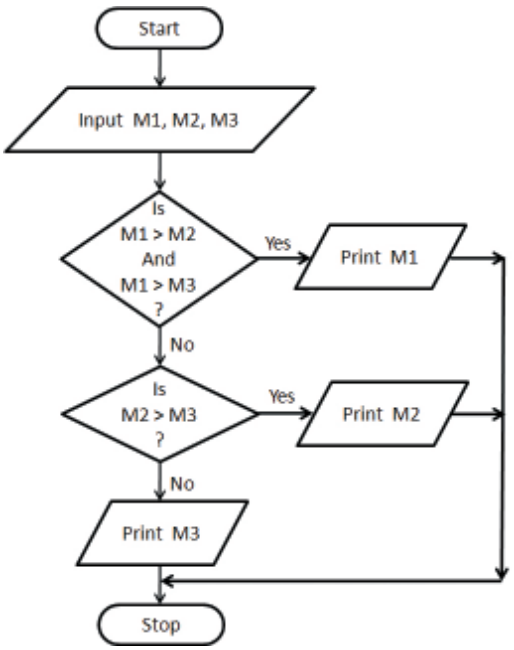


Fig. 3.11 : Flowchart to find the largest of three numbers

Let us do

1. *Develop an algorithm and draw the flowchart to check whether a given number is even or odd.*
2. *Design an algorithm and flow chart to input a day number and display the name of the day. (For example, if 1 is the input, the output should be Sunday. If it is 2, the output should be Monday. If the number is other than 1 to 7, the output should be "Invalid data").*
3. *Based on the evaluation system for standard X, develop an algorithm to accept a score out of 100 and find the grade.*

Now, consider a case in which some task is to be performed in a repeated fashion. Suppose we want to print the first 100 natural numbers. How do we do it? We know that the first number is 1. It should be printed. The next number is obtained by adding 1 to the first number. Again it should be printed. It is clear that the two tasks - printing a number and adding 1 to it - are to be executed repeatedly. The execution should be terminated when the last number is printed. Let us develop the algorithm for this.

Example 3.5: To print the numbers from 1 to 100

Step 1: Start
Step 2: N = 1
Step 3: Print N
Step 4: N = N + 1
Step 5: If N <=100 Then Go To Step 3
Step 7: Stop

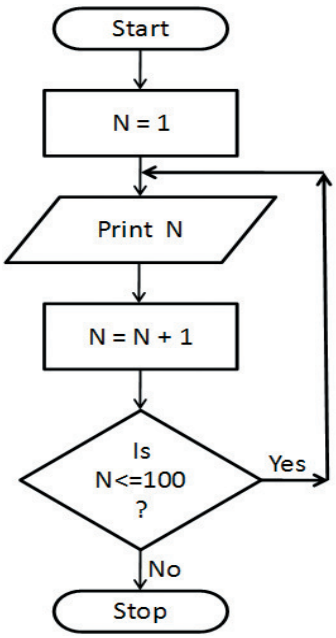


Fig. 3.12 : Flowchart to print numbers from 1 to 100

In the algorithm given as Example 3.5, a condition is checked at step 5. If the condition is found true, the flow of control is transferred back to step 3. So the steps 3, 4 and 5 are executed repeatedly as long as the condition is true. We will say that a loop is formed here. Steps 3, 4 and 5 constitute a loop. The control comes out of the loop only when the condition becomes false. The flowchart of this algorithm is shown in Figure 3.12.

The above algorithm can be simplified as follows:

Step 1: Start
Step 2: N = 1

Step 3: Repeat Steps 4 and 5 While (N<=100)
Step 4: Print N
Step 5: N = N + 1
Step 6: Stop

Note that in step 3, the words "**Repeat**" and "**While**" are used to construct a loop. The statements (step numbers) that need repeated execution is specified with the word "**Repeat**" and the condition is given with "**While**". As the algorithm looks slightly different, the flow chart also differs slightly as in Figure 3.13. The execution style of a loop is shown in Figure 3.14.

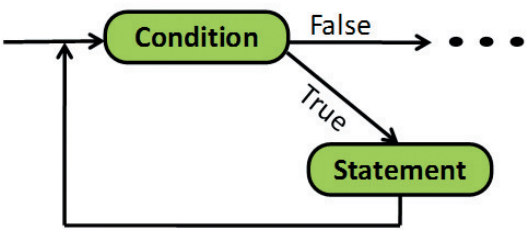


Fig. 3.14 : Looping construct

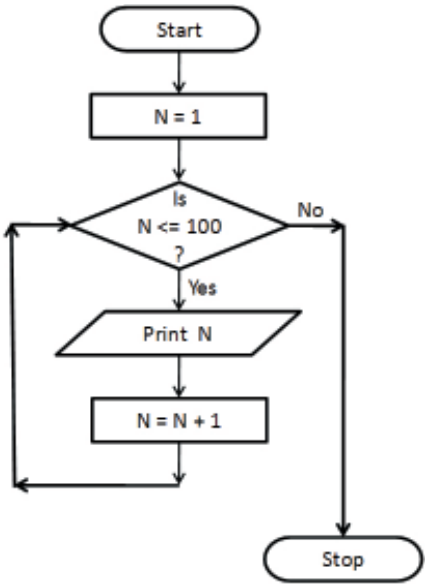


Fig. 3.13 : Flowchart to print numbers from 1 to 100

A loop has four elements. Obviously, one of them is the ***condition***. We know that at least one variable will be used to put up a condition and let us call it loop control variable. Before the condition being checked, the loop control variable should get a value. It is possible through input or assignment. Such an instruction is called **initialisation instruction** for the loop. The third element, called **update instruction**, changes the value of the loop control variable. It is essential; otherwise the execution of the loop will never be terminated. The fourth element is the **loop body**, which is the set of instructions to be executed repeatedly. The flowchart shown in Figure 3.15 depicts the working of looping structure.

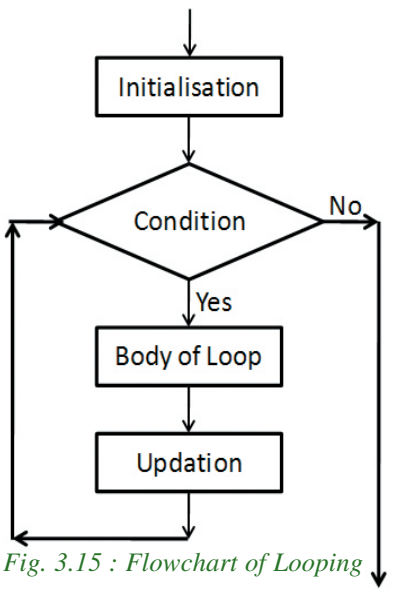


Fig. 3.15 : Flowchart of Looping

The initialisation instruction will be executed first and then the condition will be checked. If the condition is true, the body of the loop will be executed followed by the update instruction. After the execution of the update instruction, the condition will be checked again. This process will be continued until the condition

becomes false. The loop that checks the condition before executing the body is called entry-controlled loop. There is another style of looping construct. In this case, the condition will be checked only after the execution of loop-body and update instruction. Such a loop is called exit-controlled loop.

Example 3.6: To print the sum of the first N natural numbers

Here we have to input the value of N. The sum of numbers from 1 to the input number N is to be found out. Let S be the variable to store the sum. Figure 3.16 shows the flowchart of this algorithm

Step 1: Start
Step 2: Input N
Step 3: A = 1, S = 0
Step 4: Repeat Steps 5 and 6 While (A <= N)
Step 5: S = S + A
Step 6: A = A + 1
Step 7: Print S
Step 8: Stop

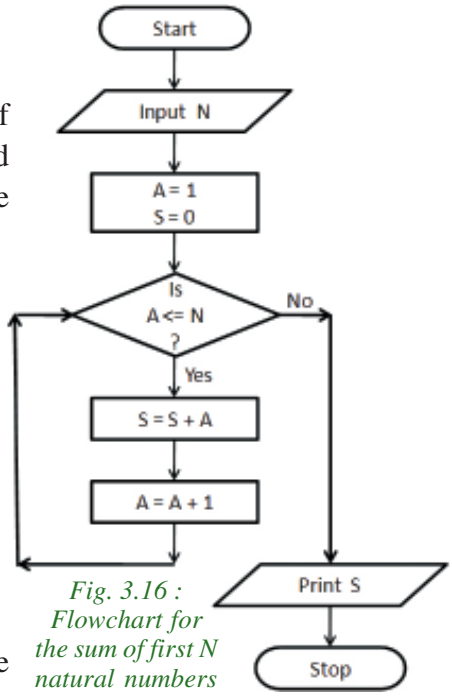


Fig. 3.16 : Flowchart for the sum of first N natural numbers

This algorithm uses an entry-controlled loop. In the next example we can see an algorithm that uses exit-controlled loop for problem solving.

Example 3.7: To print the first 10 multiples of a given number

Step 1: Start
Step 2: Input N
Step 3: A = 1
Step 4: M = A × N
Step 5: Print M
Step 6: A = A + 1
Step 7: Repeat Steps 4 to 6 While (A <= 10)
Step 8: Stop

This algorithm and the corresponding flowchart shown in Figure 3.17 contain a loop in which the condition is checked only after executing the body. Table 3.1 shows comparison between entry controlled loops and exit controlled loops.

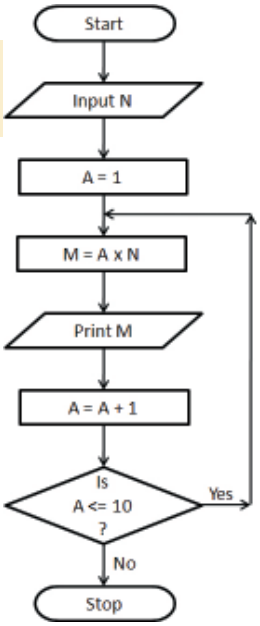


Fig. 3.17 : Flowchart for the first 10 multiples of a number

Entry controlled loop	Exit controlled loop
• Condition is checked before the execution of the body	• Condition is checked after the execution of the body
• Body may never be executed.	• Body will surely be executed at least once.
• Suitable when skipping of the body from being executed is required	• Suitable when normal execution of the body is to be ensured.

Table 3.1 : Comparison of loops

Let us practice with more algorithms and flowcharts for solving the problems that are given as learning activities.

Let us do

Develop an algorithm and draw the flowcharts for the following problems:

1. *To print all even numbers below 100 in the descending order.*
2. *To find the sum of odd numbers between 100 and 200.*
3. *To print the multiplication table of a given number.*
4. *To find the factorial of a number.*
5. *To input a number and check whether it is prime or not.*

3.3.3 Coding the program

Once we have developed the skill to design algorithms and flowcharts, the next step in programming is to express the instructions in a more precise and concise notation. That is, the instructions are to be expressed in a programming language. The process of writing such program instructions to solve a problem is called **coding**. Text editor programs are available to write the code in computer.

A language is a system of communication. We communicate our ideas and emotions to one another through natural languages such as English, Malayalam, etc. Similarly, a computer language is used to communicate between user and computer. A human being who writes a computer program is to be familiar with a language that is understandable to the computer also. We have already seen that computer knows only the binary language which is very difficult for human beings to understand and use. As we saw in Chapter 2, we can use a human friendly language, known as High Level Language (HLL) that looks similar to English. Again there is a facility of using language processors to convert or translate the program

written in HLL into machine language. The program written in any HLL is known as **source code**.

Hence, to be a programmer we have to be well-versed in any HLL such as BASIC, COBOL, Pascal, C++ etc. to express the instructions in a program. Each language has its own character set, vocabulary, grammar (we call it syntax) to write programs. Once the program is written using a language, it should be saved in a file (called source file), and then proceed to the next phase of programming.

3.3.4 Translation

While selecting a language for developing source code, certain criteria such as volume of data, complexity in process, usage of files, etc. are to be considered. Once a language is selected and the source code is prepared, it should be translated using the concerned language processor. **Translation** is the process of converting a program written in high level language into its equivalent version in machine language. The compiler or interpreter is used for this purpose. During this step, the syntax errors of the program will be displayed. These errors are to be corrected by opening the file that contains the source code. The source code is again given for compilation (translation). This process will be continued till we get a message such as "No errors or warnings" or "Successful compilation". Now we have a program fully constituted by machine language instructions. This version of the source code is known as **object code** and it will be usually stored in a file by the compiler itself.

Fig. 3.18 : Translation Process

Once the object code is obtained it should be present in the system as long as you want the program to be used.

3.3.5 Debugging

Debugging is the stage where programming errors are discovered and corrected. As long as computers are programmed by human beings, the programs will be subject to errors. Programming errors are known as 'bugs' and the process of detecting and correcting these errors is called **debugging**. In general there are two types of errors that occur in a program - syntax errors and logical errors. **Syntax errors** result when the rules or syntax of the programming language are not followed. Such program errors typically involve incorrect punctuation, incorrect word sequence, undefined term, or illegal use of terms or constructs. Almost all language processors detect syntax errors when the program is given for translation. They print error messages that include the

line number of the statement having errors and give hints about the nature of the error. In the case of interpreters, the syntax errors will be detected and displayed during execution. The programmer's efficiency in using the language decides the time and effort for the debugging process. The object program will be generated only if all the syntax errors are rectified.

The second type of error, named **logical error**, is due to improper planning of the program's logic. The language processor successfully translates the source code into machine code if there are no syntax errors. During the execution of the program, computer actually follows the program instructions and gives the output as per the instructions. But the output may not be correct. This is known as logical error. When a logical error occurs, all you know is that the computer is not giving the correct output. The computers do not tell us what is wrong. It should be identified by the programmer or user. In order to determine whether or not there is a logical error, the program must be tested. So, let us move on to the next stage of programming.

3.3.6 Execution and testing

As we have seen in the previous section, the program is said to be error-free only when logical errors are also rectified. Hence when the compiled version of the program is formed, it should be executed for testing. The purpose of testing is to determine whether the results are correct. The testing procedure involves running the program to process the test data that will produce 'known results'. That is, the operations involved in the program should be done manually and the output thus obtained should be compared with the one given by the computer. The accuracy of the program logic can be determined by this testing. While selecting the test data, we should ensure that all aspects of the program logic will be tested. Hence the selection of proper test data is important in program testing.

Till now, we have discussed incorrect outputs due to incorrect logic. But there is a chance of another type of error, which will interrupt the program execution. This may be due to the inappropriate data that is encountered in an operation. For example consider an instruction A= B/C. This statement causes interruption in execution if the value of C happens to be zero. In such a situation, the error messages may be displayed by the error-handling function of the language. These errors are known as **Run-time error**. These errors can be rectified by providing instructions for checking the validity of the data before it gets processed by the subsequent instructions in the program.

3.3.7 Documentation

A computerised system cannot be considered to be complete until it is properly documented. In fact documentation is an on-going process that starts in the problem-study phase of the system and continues till its implementation and operation. We can write comments in the source code as part of documentation. It is known as **internal documentation**. It helps the debugging process as well as program modification at a later stage. The logic that we applied in the program may not be remembered when we go through our own program at a later stage. Besides, the program written by one person may need to be modified by some other person in future. If the program is documented, it will help to understand the logic we applied, the reason why a particular statement has been used and so on. However, the documentation part of the program will not be considered by the language processor when you give the program for translation.

Writing comments in programs is only a part of documentation. Another version of documentation is the preparation of system manual and user manual. These are hard copy documents that contain functioning of the system, its requirements etc. and the procedure for installing and using the programs. While developing software for various applications, these manuals are mandatory. This kind of documentation is known as **external documentation**.

Now you have analysed the problem, derived the logic of the solution, expressed in a flow chart, developed the code in a programming language, translated it after removing the syntax errors, checked the accuracy of the output after removing all the possible logical and run-time errors, and we have documented the program.

Check yourself

1. What is an algorithm?
2. Pictorial representation of algorithm is known as _____.
3. Which flow chart symbol is always used in pair?
4. Which flow chart symbol has one entry flow and two exit flows?
5. Program written in HLL is known as _____.
6. What is debugging?
7. What is an object code?

3.4 Performance evaluation of algorithms

We have developed algorithms for solving various problems. You may think that some of these problems would have been solved by following a different logic. Of course, it is true that the same problem can be solved by different sets of instructions. But an efficient programmer is the one who develops algorithms that require minimum computer resources for execution and give results with high accuracy in lesser time. The performance of an algorithm is evaluated based on the concept of time and space complexity. The algorithm which will be executed faster with minimum amount of memory space is considered as the best algorithm for the problem.

Algorithm-1		Algorithm-2	
Step 1:	Start	Step 1:	Start
Step 2:	Input A, B, C	Step 2:	Input A, B, C
Step 3:	S = A + B + C	Step 3:	S = A + B + C
Step 4:	Avg = S / 3	Step 4:	Avg = (A + B + C) / 3
Step 5:	Print S, Avg	Step 5:	Print S, Avg
Step 6:	Stop	Step 6:	Stop

Table 3.2 : Algorithms to find the sum and average of three numbers

Let us compare the two algorithms given in Table 3.2, developed to find the sum and average of three numbers. The two algorithms differ in step 4. Algorithm-2 uses two steps (steps 3 and 4) for addition operation on the same data. Naturally, that algorithm will take more time for execution than Algorithm-1. So Algorithm-1 is better for coding.

Now let us take another case, where comparison operations are involved for the selection of a statement. We have already discussed an algorithm to find the largest among three numbers in Example 3.4. The two algorithms given in Table 3.3 can also be used for solving the same problem.

Algorithm-1	Algorithm-2
Step 1: Start	Step 1: Start
Step 2: Input M1, M2, M3	Step 2 Input M1, M2, M3
Step 3: If M1>M2 And M1>M3 Then	Step 3: If M1 > M2 Then
Step 4: Print M1	Step 4: Big = M1
Step 5: If M2 > M1 And M2 > M3 Then	Step 5: Else
Step 6: Print M2	Step 6: Big = M2
Step 7: If M3 > M1 And M3 > M2 Then	Step 7: If M3 > Big Then Big = M3
Step 8: Print M3	Step 8: Print Big
Step 9: Stop	Step 9: Stop

Table 3.3 : Algorithms to find the largest among three numbers

The algorithm in Example 3.4 has three comparison operations and one logical operation altogether. All these operations are to be carried out only when the largest value is in M3 (the third variable). To identify the speed of execution in each case, let us assume that 1 second is required for one comparison operation. We can see that fastest result will be in 3 seconds and slowest in 4 seconds. So the average speed is 3.5 seconds.

Now let us analyse Algorithm-1 in Table 3.3. There are three **If** statements, each with three comparison operations. If we follow the assumptions specified above, we can see that the result will be obtained in 9 seconds, irrespective of the values in the variables. So the average speed is 9 seconds. But the Algorithm-2 in Table 3.3 uses two If structures. The algorithm shows that whatever be the values in the variables, the time required for comparison will be 2 seconds. Thus the average speed is 2 seconds. So, we can say that the Algorithm-2 is better than the other two.

Let us consider one more case where loop is involved. The two algorithms given in Table 3.4 find the sum of all even numbers and sum of all odd numbers between 100 and 200.

Algorithm-1	*Algorithm-2*
Step 1: Start	Step 1: Start
Step 2: N = 100, ES = 0	Step 2: N = 100, ES = 0, OS = 0
Step 3: Repeat Steps 4 to 6 While (N <= 200)	Step 3: Repeat Steps 4 to 8 While (N <= 200)
Step 4: If Remainder of N/2 = 0 Then	Step 4: If Remainder of N/2 = 0 Then
Step 5: ES = ES + N	Step 5: ES = ES + N
Step 6: N = N + 1	Step 6: Else
Step 7: Print ES	Step 7: OS = OS + N
Step 8: N = 100, OS = 0	Step 8: N = N + 1
Step 9: Repeat Steps 10 to 12 While (N <= 200)	Step 9: Print ES
Step 10: If Remainder of N/2 = 1 Then	Step 10: Print OS
Step 11: OS = OS + N	Step 11: Stop
Step 12: N = N + 1	
Step 13: Print OS	
Step 14: Stop	

Table 3.4 : Algorithms to find sum of even and odd numbers

Algorithm-1 uses two loops. Obviously, time taken will be double for the intialisation, testing and updation of loop control variable compared to Algorithm-2. From the table, it is clear that Algorithm-2 is better and efficient. So, think divergently and differently to develop logic for solving problems.

Let us sum up

Program is a sequence of instructions written in a computer language. The process of programming proceeds through some stages. Preparation of algorithms and flowcharts help develop the logic. The program written in HLL is known as source code and it is to be converted into machine language. The resultant code is known as object code. The errors occurred in a program has to be removed through a process known as debugging. The translated version is executed by the computer. Proper documentation of the program helps us to modify it at a later stage. While solving problems different logics may be applied, but the performance is measured in terms of time and space complexity.

Learning outcomes

After the completion of this chapter the learner will be able to

- explain various aspects of problem solving.
- develop algorithms for solving problems.
- draw flowcharts to ensure the correctness of algorithms.
- select the best algorithm for solving a problem.

Sample questions

Very short answer type

1. What is an algorithm?
2. What is the role of a computer in problem solving?
3. What is the use of connector in a flow chart?
4. What do you mean by logical errors in a program?

Short answer type

1. What is a computer program? How do algorithms help to write programs?
2. Write an algorithm to find the sum and average of 3 numbers.
3. Draw a flowchart to display the first 100 natural numbers.
4. What are the limitations of a flow chart?
5. What is debugging?
6. What is the need of documentation for a program?

Long answer type

1. What are the characteristics of an algorithm?
2. What are the advantages of using a flowchart?
3. Briefly explain different phases in programming.