5 Introduction to Problem Solving

In This Chapter

5.1 INTRODUCTION

There is a famous quote by Steve Jobs, which says, *"Everyone in this country should learn to program a computer, because it teaches you to think"*.

This quote itself is proof-enough to say that in order to program a solution for a problem, you should think in a specific way. And this is what we are going to talk about in this chapter.

In this chapter, you will learn about problem solving, *i.e.*, analysing a problem, designing algorithms using tools like flowcharts and pseudocode and problem solving using decomposition. So, let us begin.

5.2 PROBLEM SOLVING CYCLE

Programs are not quick creations. In order to create efficient and effective programs, you should adopt a proper problem solving methodology and use appropriate techniques. In fact, problem solving methods follow a cycle – the *problem solving cycle*. In the coming lines, we are going to discuss the same.

Broadly problem solving requires *four* main steps :

1. Identify and analyse the problem.
2. Find its solution and Develop algorithm of the solution.
3. Code the solution in a programming language.
4. Test and Debug the coded solution.

And finally **implement and maintain** it.

The above mentioned steps are four major steps in a problem solving cycle. Each step contains many sub-steps. Let us talk about these sub-steps in order to understand the problem solving cycle. The aim of a problem solving cycle is to create a working program for the solution.

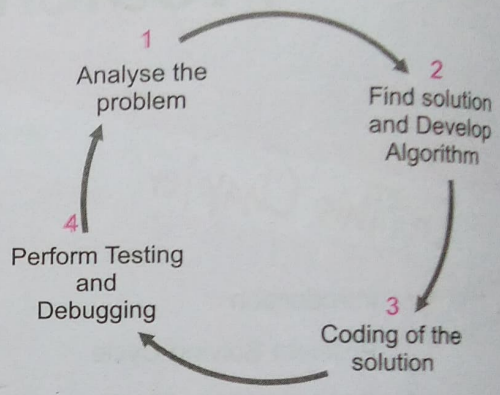


Figure 5.1 Steps in Problem solving cycle.

The sub-steps of a problem solving cycle to create a working program are :

Analyze the problem	1.	*Understand the problem well*	It is very important to understand the problem minutely because sometimes the problem is not that appears but something else that is causing it.
	2.	*Analyze the problem*	Problem analysis is very important as the outcome of this will convert to the success of final solution. While analyzing, ■ identify processing components ■ identify the relationships among processing components.
Developing the Algorithm	3.	*Think of possible solutions*	As per the problem analysis, think of possible solutions by trying this : ■ think of different ideas for solutions ■ check your ideas for aptness ■ finally, zero on most appropriate solution.
	4.	*Follow Modular approach while designing most appropriate solution*	Many small logically related modules or functions must be preferred over a big problem. It is called *Modular approach* wherein we divide a big program into many smaller and more manageable and understandable modules. Design the final program by ■ deciding step by step solution ■ breaking down solution into simple steps.
	5.	*Identify operations for solution*	Program coding involves identification of constituent operations required to get the desired output. One must decide about the minimum but simple operations required. Identify : ■ minimum number of inputs required ■ arithmetic and logical operations required for solutions ■ simple and efficient data structures suiting the need.

Coding	6.	*Code program using appropriate control structures*	The next step is to code the program as per finding of previous step. Coding is the technical word for writing the program. This step is to translate the algorithm into a programming language. You should use most appropriate control structures out of available options. Thus, it is important to know the working of different control structures and their suitability in different situations. ■ Use appropriate control structures such as conditional or looping control structures. ■ Think program's efficiency in terms of speed, performance and effectiveness.
Testing and Debugging	7.	*Test and Debug your program*	Testing is the process of finding errors in a program and debugging is the process of correcting errors found during the testing process. Thus, this phase involves : ■ finding errors in it. ■ rectifying the errors.
	8.	*Complete your documentation*	Documentation is intended to allow another person or the programmer at later date, to understand the program. Documentation might also consist of a detailed description of what the program does and how to use the program.
Implement and Maintain	9.	*Implement your code*	After testing and documentation, implement your program for actual use on site. Now, the real users can use your programs.
	10.	*Maintain your program*	Maintaining programs involves modifying the programs to remove previously undetected errors, to enhance the program with different features or functionality, or keep the program up-to-date as government regulations or company policies change.

5.2.1 Problem Solving using Decomposition

The previous section talked about problem solving cycle. The first five steps of problem solving cycle are very important as rest of the steps are based on them.

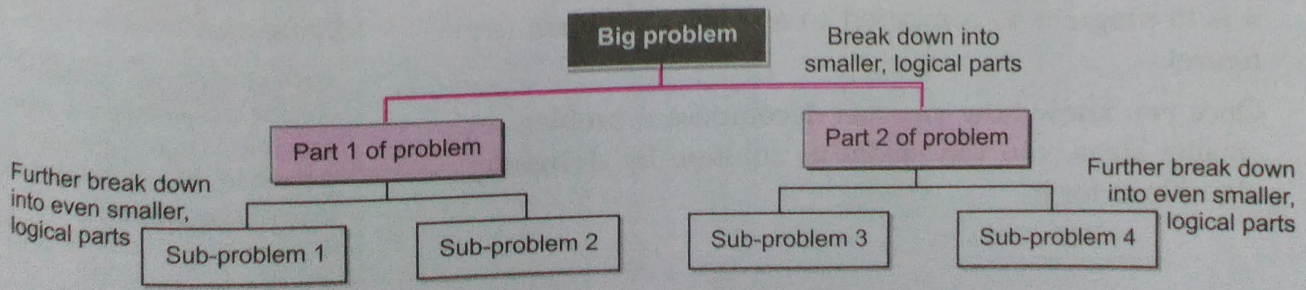
The first five steps involve understanding the problem, analyzing it, and thinking of possible solution with sub-module and operations in it. All this is effectively carried out using decomposition. Let us know what decomposition means.

DECOMPOSITION

The process of breaking down a big or complex problem into a set of smaller sub-processes in order to understand a problem or situation better, is known as **decomposition**.

Decomposition is the process of breaking down a big or complex problem into a set of smaller sub-processes to allow us to describe, understand, or execute the problem better. Decomposition involves :

- Dividing a task into a sequence of subtasks.
- Identifying elements or parts of a complex system.



Consider some examples of decomposition :

- **Everyday example.** Making cookies is a complex task that can be broken down into smaller, simpler tasks such as mixing up the dough, forming into shapes via cookie cutters, and baking.
- **Academic example.** Writing an essay is a complex task that can be broken down into smaller tasks such as developing a thesis, gathering evidence, and creating a bibliography page.
- **Engineering example.** Designing a solution to construct a bridge by considering site conditions, technology available, technical capability of the contractor, foundation, etc.
- **Computer Science example.** Writing a computer program/software by determining a well-defined series of smaller steps (mostly in the form of modules and functions) to solve the problem or achieve a desired outcome.

EXAMPLE 1 *Decompose the task of creating mobile app.*

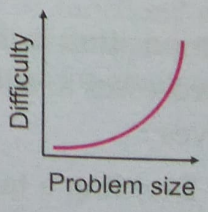
SOLUTION To decompose the task of creating a mobile app, we would need to know the answer to a series of smaller problems :

- what kind of app is to be created.
- who the target audience for the app is.
- what the user interface of the app will be (what all screens, type of input etc.).
- what the app's graphics will look like.
- what audio will be included.
- what software/ platform will be used to build the app.
- how the user will navigate the app.
- what additional services will be required for the app, *e.g.*, database etc.
- how the app will be tested.

This list has broken down the complex problem of creating an app into much simpler problems that can now be worked out.

Need for Decomposition

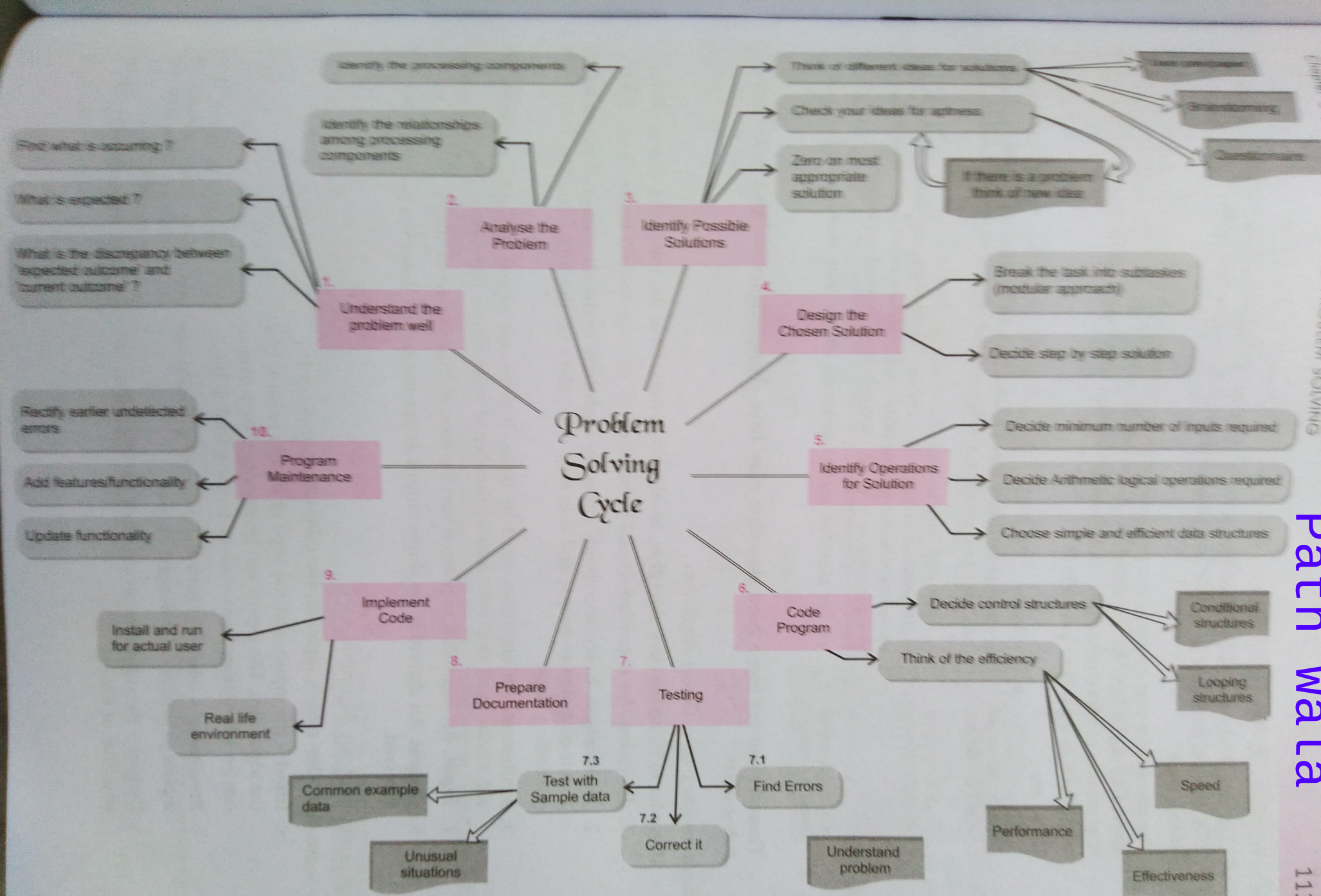
Decomposition is the process of breaking a large problem into more manageable sub-problems. It is very important to decompose a problem into smaller sub-problems, reason being that large problems are disproportionately harder to solve than smaller problems. It's much easier to write two 500-line programs than a single 1000-line program. Larger a problem is, harder and more difficult it is to program as compared to a smaller problem (*see* figure).



Once you know how you can decompose a problem in smaller steps, you can create its solution by designing algorithms for it.

NOTE

Decomposing a problem into smaller sub-problems is important because large problems are disproportionately harder to solve than smaller problems.



Problem Solving Methodology

5.3 DESIGNING ALGORITHMS

Computers are essentially problem solving devices. Computers solve these problems by dividing the problems in simple and small steps that are expressed in the form of computer instructions.

The set of rules that define how a particular problem can be solved in finite number of steps is known as algorithm. An algorithm is composed of a finite set of steps, each of which may require one or more operations. But there are certain constraints to be placed on the type of operations as algorithm can include.

These are :

Each operation must be definite	*i.e.*, it must be clearly defined what should be done. For instance, '$x = 6/0$', 'add 3 or 8 to a' are not permitted as these operations are not clearly defined.
Each operation must be effective	*i.e.*, each step must be such that it can be done using pencil and paper in a finite amount of time. *For example*, arithmetic on integers is effective operation, whereas arithmetic on real numbers is not since some values may be expressible only by an infinitely long decimal expansion.
Each operation must be finite	*i.e.*, the algorithm should terminate after a finite number of operations.

Thus, we can say that a good algorithm must have characteristics as listed in the following sub-section.

Characteristics of a Good Algorithm

In order to be an effective algorithm, an algorithm must have the following characteristics :

1. **Precision.** An algorithm should be precise, *i.e.*, its steps should be precisely defined.
2. **Uniqueness.** Every step should uniquely contribute to the algorithm. It means that the result of a step is unique and it is only dependent on the input and the result of the preceding steps.
3. **Finiteness.** An algorithm must be finite. It must not repeat the steps endlessly. The algorithm must stop after a finite number of instructions have been executed.
4. **Input.** An algorithm requires a specific type of input to work upon.
5. **Output.** An algorithm produces output as per the stated objectives and the input it has received.

Components of an Algorithm

An algorithm clearly identifies the following components :

- **Input** – the inputs and the type of inputs required by the algorithm. The inputs can be provided by a user or can be self-obtained too, such as reading from a file.
- **Processing** – what and how the processing would use inputs to produce the desired output.
- **Output** – the output expected from the algorithm ; the objective of the algorithm.

NOTE

One must be able to identify where inputs, processing and outputs are taking place within an algorithm.

A program is the expression of an algorithm in a programming language. Thus, the success of a program depends upon the algorithm. Therefore, the logic of the problem must be clearly expressed in algorithm. The logic of the problem can be expressed in various manners. One of the most preferred methods is the graphical method of representing the problem's solution, which is known as flowchart.

Another useful tool for designing algorithms is *pseudo-code*. In the coming subsections, we are talking about both these tools.

5.3.1 Flowcharts

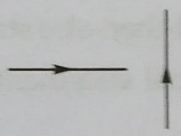
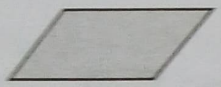
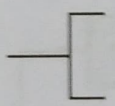
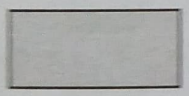
A flowchart is a pictorial representation of step by step solution of a problem.

A flowchart not only pictorially depicts the sequence in which instructions are carried out in an algorithm but also is used as an aid in developing algorithms. One must be familiar with such an important tool used in programming. This section briefly discusses the technique of flow charting.

FLOWCHART

A **flowchart** is a pictorial representation of step by step solution of a problem.

There are various flowchart symbols that carry different messages and are used for different purposes. These symbols are shown below :

Symbol	Purpose	Symbol	Purpose
	Start/Stop		Flow of control symbol
	Input/Output		Annotation
	Processing		Connector
	Decision Box		

Following section illustrates the working and use of flow charts along with algorithm development.

Writing Algorithms

To write algorithms, such a language should be used that is close enough to the programming language(s) (in which the programs are to be written) so that a hand translation is relatively easy to accomplish. Thus, we have chosen a language for algorithms that resembles our programming language Python.

Let us discuss certain rules for writing algorithms.

Identifiers

Identifiers are the names given to various components of a program by the programmer *e.g.*, to variables that hold values, to functions, modules etc.

While choosing names for identifiers, make sure that these are meaningful and not unnecessarily long or short names.

Assignment

The assignment of values to variables is done through assignment statement as :

```
variable ← <expression>
```

e.g., `A ← 10`

In an algorithm, the left arrow (←) denotes the act of assigning the value of its right-hand side to the variable on its left. Some people take liberty and use assignment operator to assign values to variables in algorithms.

Sequence

The steps of an algorithm are executed in the sequence of top to bottom. One after another so steps must be listed in the correct order.

Selection/Conditional Statements

A conditional statement in an algorithm takes the following form :

```
if condition :
    statement      # block 1
```

OR

```
if condition :
    statement      # block 1
else
    S2             # block 2
```

There may be one or more if-then-else statements embedded in another if-then-else statement. Figure 5.1 depicts conditional statements pictorially.

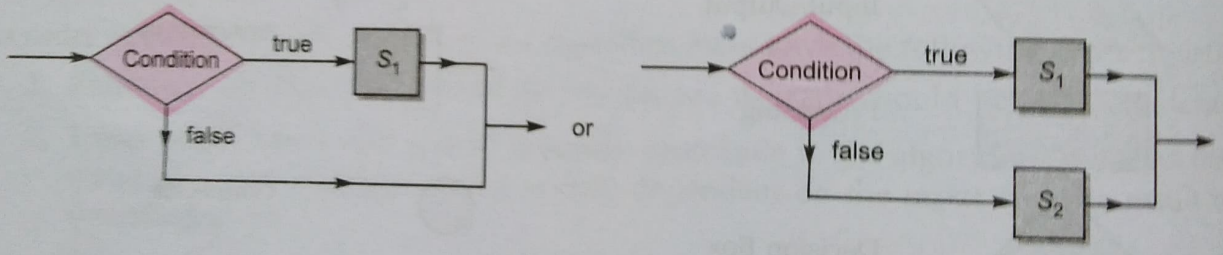


Figure 5.1 The conditional **If** statements

Repetition – Looping Statement

A looping statement (also called iteration statement) lets you repeat a set of statements depending upon a condition. To accomplish iteration, the **for loop** and **while loop** are used.

for looping statement

```
for item in sequence :
    : St                    # block of statements
:
```

St represents the set of statements to be repeated for each item in the sequence. When the statements block gets executed for each item in the sequence, the for-loop stops.

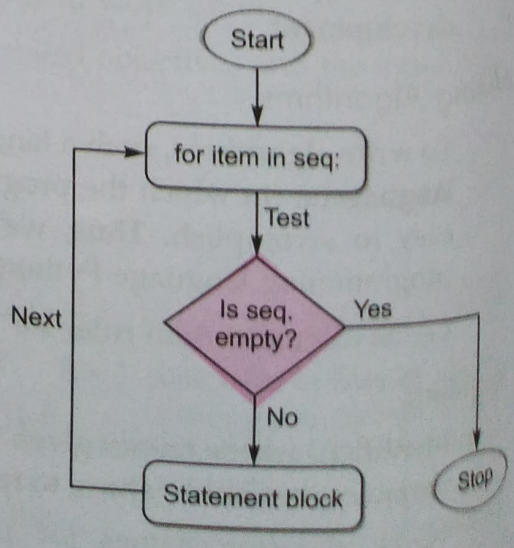


Figure 5.2 (*a*) for loop process.

while looping statement

```
while condition :
    :    St        # block of statements
:
```

St represents the set of statements to be repeated. The *while* iteration statement tests the condition before entering into the loop. Thus, if the condition is false even before entering into the loop, the while-loop will never get executed.

Let us now consider some examples of algorithms and flowcharting.

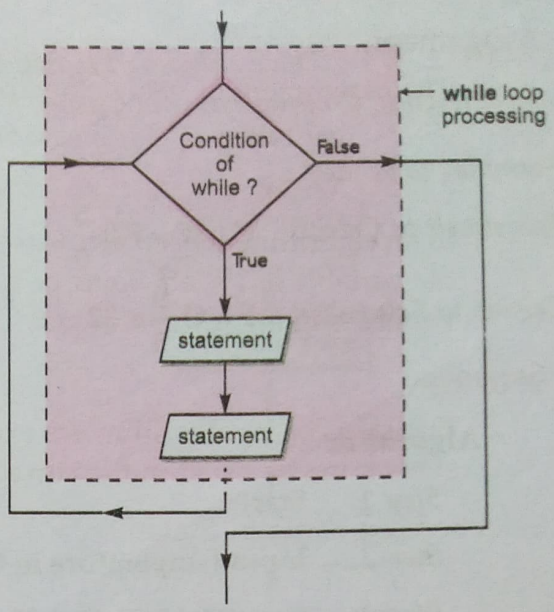


Figure 5.2 (*b*) while loop processing

EXAMPLE 2 *Draw a flowchart to add two numbers, along with algorithm in simple English.*

SOLUTION

Algorithm

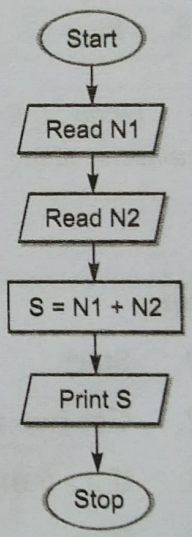
Step 1. Start

Step 2. Get two numbers N1 and N2

Step 3. S ← N1 + N2

Step 4. Print the result S

Step 5. Stop



EXAMPLE 3 *Draw a Flowchart to find Area and Perimeter of Rectangle, along with algorithm in simple English.*

L : Length of Rectangle, ***B*** *: Breadth of Rectangle*

AREA : Area of Rectangle, ***PERIMETER*** *: Perimeter of Rectangle*

SOLUTION

Algorithm

Step 1. Start

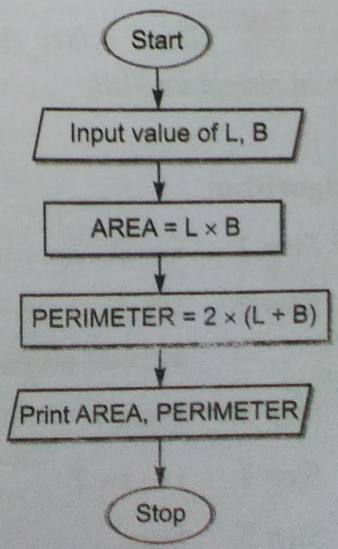
Step 2. Input Side-Length & Breadth say *L, B*

Step 3. AREA ← $L \times B$

Step 4. PERIMETER ← $2 \times (L + B)$

Step 5. Print AREA, PERIMETER

Step 6. Stop

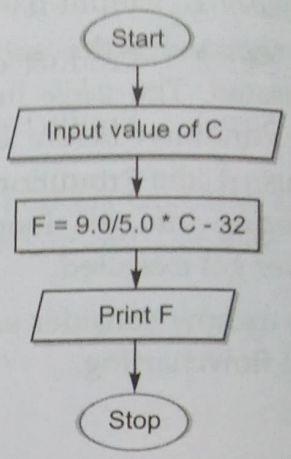


EXAMPLE 4 *Draw a flowchart to convert temperature from* **Celsius** *to* **Fahrenheit** *along with algorithm in simple English. C : temperature in Celsius, F : temperature Fahrenheit*

Formulas to be used :

Fahrenheit to Celsius : $C = (F - 32).\frac{5}{9}$;

Celsius to Fahrenheit : $F = C.\frac{9}{5} + 32$



SOLUTION

Algorithm

Step 1. Start

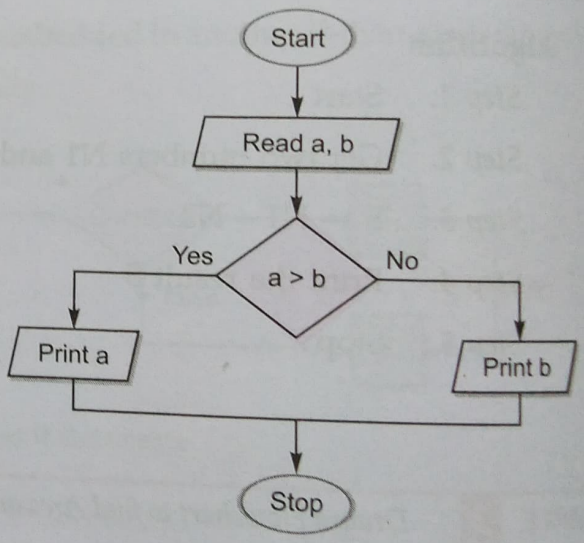
Step 2. Input temperature in Celsius in variable C

Step 3. $F \leftarrow (9.0 / 5.0 \times C) + 32$

Step 4. Print Temperature in Fahrenheit F

Step 5. Stop

EXAMPLE 5 *Draw a flowchart to determine the larger of two numbers, along with algorithm in simple English.*



SOLUTION

Algorithm

Step 1. Start

Step 2. Get two numbers *a* and *b*

Step 3. If $a > b$ then print *a* else print *b*

Step 4. Stop

EXAMPLE 6 *Draw a flow chart to print even numbers from 2 to* 10^4 *using a loop approach, along with algorithm in simple English.*

SOLUTION

Algorithm

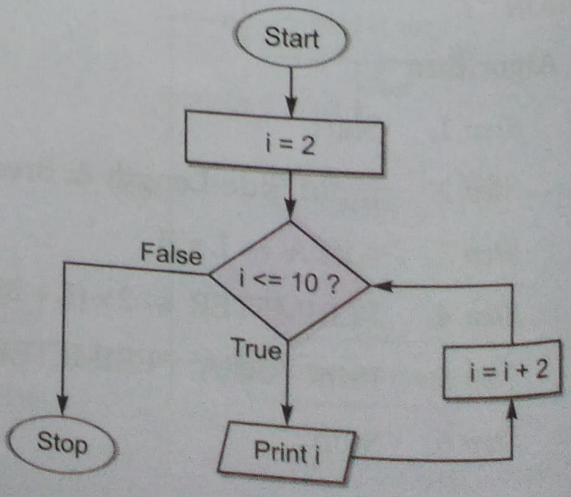
Step 1. $i \leftarrow 2$

Step 2. While $i < 10$
Repeat steps 3 through 4

Step 3. Print *i*

Step 4. $i \leftarrow i + 2$

Step 5. Stop

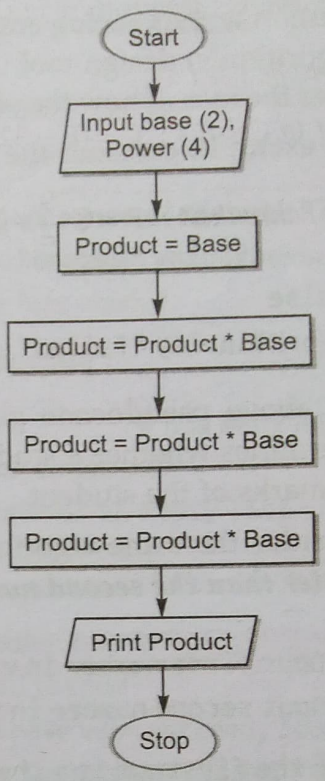


EXAMPLE 7 *Draw a flow chart to calculate* 2^4, *along with algorithm in simple English.*

SOLUTION

Algorithm

Step 1. Input Base (2), Power (4)
Step 2. Product ← Base
Step 3. Product ← Product * Base
Step 4. Product ← Product * Base
Step 5. Product ← Product * Base
Step 6. Print Product
Step 7. Stop

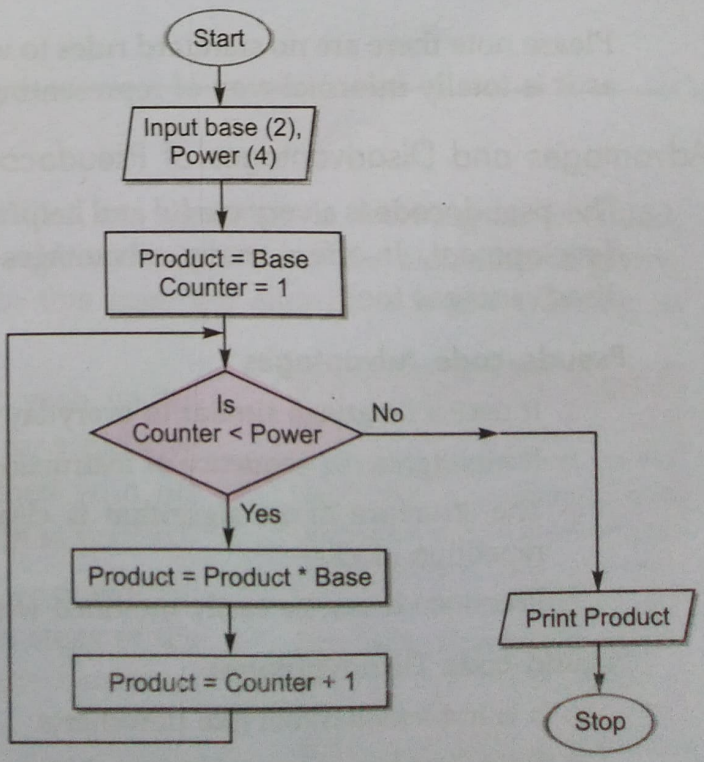


EXAMPLE 8 *Draw a flow chart to calculate* 2^4 *using a loop approach, along with algorithm in simple English.*

SOLUTION

Algorithm

Step 1. *Input* Base (2), Power (4)
Step 2. Product ← Base
Step 3. Counter ← 1
Step 4. *While* (Counter < Power)
Repeat steps 4 through 6
Step 5. Product ← Product * Base
Step 6. Counter ← Counter + 1
Step 7. Print Product
Step 9. Stop



5.3.2 Pseudocode

Pseudocode is an informal language that helps programmers describe steps of a program's solution without using any programming language syntax. Pseudocode is a "text-based" detail (algorithmic) design tool. A pseudocode creates an outline or a rough draft of a program that gives the idea of how the algorithm works and how the control flows from one step to another.

For example, consider the following pseudocode :

```
If student's marks is greater than or equal to 50
    display "passed"
else
    display "failed"
```

The above pseudocode gives you an idea how a program determines whether a student has *passed* or *failed* comparing the marks of the student.

NOTE

Pseudocode is an informal way of describing the steps of a program's solution without using any strict programming language syntax or underlying technology considerations.

Consider the same algorithm that we talked in example 5 (***determine if the first number is greater than the second number or not***). The pseudocode of this algorithm can be like :

```
Input first number in variable firstnum.
Input second number in variable secondnum

if the firstnum is > the secondnum
    display 'the first number IS greater than second number'.
else
    display 'the first number IS greater than second number'.
```

Please note there are no standard rules to write a pseudocode as it is totally informal way of representing an algorithm.

NOTE

In pseudocode, usually the instructions are written in upper-case, variables in lowercase and messages in sentence case.

Advantages and Disadvantages of Pseudocode

The pseudocode is a very useful and helpful tool in algorithm development. It offers many advantages and it has some disadvantages too.

Pseudo-code Advantages

- It uses a language similar to everyday English, thus is easy to understand.
- It highlights the sequence of instructions.
- The structure of an algorithm is clearly visible through pseudocode, *e.g.*, selection and repetition blocks.
- Pseudocode can be easily modified without worrying about any dependencies.

Pseudo-code Disadvantages

- It is not a visual tool like flowcharts.
- Since there is no accepted standard for pseudocode, so it varies from programmer to programmer.
- It can only be tested on paper, there is no program available to test it.
- It is an informal representation of an algorithm.

EXAMPLE 9 *Write pseudocode that converts from Fahrenheit to Celsius or from Celsius to Fahrenheit, depending on the user's choice.*

SOLUTION **Pseudocode :**

```
x = INPUT "Press 1 to convert from Fahrenheit
    to Celsius or Press 2 to convert from
    Celsius to Fahrenheit."
y = INPUT ask what number?
IF 1 is pressed
    # do f to c conversion
    C = 5/9 * (F - 32)
    PRINT output C
ELSE IF 2 is pressed
    # do c to f conversion
    F = 9/5 * (C + 32)
    PRINT output F
ELSE
    PRINT "Please enter either 1 or 2"
```

EXAMPLE 10 *Write pseudocode that lets the user type words and when they press 'x', it prints how many words the user inputted then quits program.*

SOLUTION **Pseudocode :**

```
WHILE true
    INPUT "Enter a Word"
    PRINT word
    IF word = 'x'
        PRINT number of words entered
        BREAK OUT from the loop
    ELSE
        add 1 to count
```

EXAMPLE 11 *Write pseudocode that asks how many numbers (say n), and then print all those numbers after n numbers have been entered.*

SOLUTION **Pseudocode :**

```
INPUT value of n            # how many numbers
Initialise numbercount = 0
Initialise listnum as an empty list
WHILE the numbercount <= n
    ask for a number
    add one to number count
    add number to listnum
# now the numbers have been entered, loop is over
# take listnum's numbers in item one by one
FOR item in listnum
    PRINT item
    # item now takes next number in listnum
```

5.3.3 Verifying an Algorithm

Verification of an algorithm means to ensure that the algorithm is working as intended. For example, if you have written a program to add two numbers but the algorithm is giving you the product of the two input numbers. In this case, the algorithm is not working as intended.

Then how do you verify an algorithm ? Ohh, yeah you're right – we should test it with a sample inputs for which we know the output; if the expected output matched with the output produced by the algorithm, the algorithm is verified.

This is really helpful, but for larger algorithms, we may want to verify the intermediate results too of various steps of the algorithm. And for such requirements, a useful mechanism of verifying algorithm called **Dry-Run** is used.

NOTE

The word **'algorithm'** comes from the ninth-century Arab mathematician, *Al-Khwarizmi*, who worked on 'written processes to achieve some goal.' The term 'algebra' also comes from the term *'al-jabr,'* which he introduced.

5.3.3A Dry Run

A dry run is the process of a programmer manually working through their code to trace the value of variables. There is no software involved in this process.

Traditionally, a dry run would involve a print out of the code. The programmer would sit down with a pen and paper and manually follow the value of a variable to check that it was used and updated as expected.

If a programmer found that the value is not what it should be, they are able to identify the section of code that resulted in the error.

DRY RUN

A dry run is the process of a programmer manually working through their code to trace the value of variables. There is no software involved in this process.

Characteristics of a dry run are :

- It is carried out during design, implementation, testing or maintenance.
- It is used to identify the logic errors in a code.
- It cannot find the execution errors in a code.

Trace Tables

Dry running an algorithm is carried out using trace tables where the impact of each line of the code is seen on the values of **variables** of the algorithm. As per the line of the code, the processing that takes place is applied on the variables' values.

For example, we want to calculate the double of the numbers in sequence 1, 2, 3 and print the value as 1 added to the double value :

To see Dry Run in action

Scan QR Code

```
1.  FOR number ← 1 TO 3
2.      Val ← number * 2
3.      PRINT Val + 1
```

The trace table for the above code will record the code movement and its impact on the variables, line by line :

Line number	Number	Val ← number * 2	Print Val + 1
1	1		
2		2	
3			
			3
	Loop's next pass starts		
1	2		
2		4	
3			
			5
	Loop's next pass starts		
1	3		
2		6	
3			
			7

Number assigned value 1

Val is = number *2

Val+1 is printed

So the given code's trace table will be as shown above, when the code is dry-run. The above given trace table has documented the impact of each line of the code separately.

However, you can skip this style and simply show the impact of code on variables too, e.g., as shown below :

Line numbers	Number	Val ← number * 2	Print Val + 1
1-3	1	2	3
1-3	2	4	5
1-3	3	6	7

You can choose to skip line numbers too, if you want as we have done in the example below.

NOTE

Trace tables enable the variable values in an algorithm to be recorded as the algorithm is dry run.

EXAMPLE 12 *Dry-run the code given below and show its trace table.*

```
num ← USER INPUT
count ← 0
WHILE num < 500 THEN
    num ← num * 2
    count ← count + 1
# end while
PRINT num
PRINT count
```

SOLUTION

num	count	num < 500	OUTPUT (Values printed)
16	0	TRUE	
32	1	TRUE	
64	2	TRUE	
128	3	TRUE	
256	4	TRUE	
512	5	FALSE	
			512
			5

Check Point 5.1

1. Name some useful tools for program development.
2. What is the difference between an algorithm and pseudocode ?
3. Which of the following is graphical ?
   (a) algorithm (b) flowchart
   (c) pseudocode (d) dry run
4. Which of the following useful for tracing a pseudo code ?
   (a) algorithm (b) flowchart
   (c) pseudocode (d) dry run
5. What is the use of trace table ?

5.3.4 Comparing Algorithms

To solve a problem, many a times, in fact, mostly multiple solutions are available. In other words, you may have multiple algorithms which working correctly and producing the correct, desired results. In such a case, which algorithms should you choose for coding ?

In order to decide which algorithm to choose over another, their efficiency will be the deciding factor.

Major factors that govern the efficiency of an algorithm are :

- the time it takes to find the solution and
- the resources which are consumed in the process.

For instance while buying a car, how do you choose a car from available choice? Yup, you are absolutely right; you might consider both the speed and the fuel consumption. The factor that matters the most to you, will win ☺.

Similarly, for algorithms, the *two* deciding factors are :

- **Space-wise efficiency.** How much amount of (memory) space the algorithm will take up before it terminates. For this, we consider the amount of data the algorithm uses while running.
- **Time-wise efficiency.** How much time the algorithm takes to complete. This factor is dependent on so many issues like, RAM available, programming language chosen, the hardware platform and many others.

When you have hardware platform fixed with a specific RAM size, then space efficiency becomes the deciding factor and the algorithms that takes up less space is chosen.

With this we have come to the end of this chapter. Let us quickly revise what we have learnt so far in this chapter.

LET US REVISE

- *A Flowchart is a pictorial representation of step by step solution of a problem.*
- *The logical sequence of precise steps that solve a given problem, is called Algorithm.*
- *An algorithm must be a finite series of steps; each step must be precise ; it must be efficient i.e., terminate in a reasonable amount of time, and must produce the correct results i.e., it must be effective.*
- *The process of breaking down a big or complex problem into a set of smaller sub-processes in order to understand a problem or situation better, is known as* ***decomposition****.*
- *Decomposing a problem into smaller sub-problems is important because large problems are disproportionately harder to solve than smaller problems.*
- *Pseudocode is an informal way of describing the steps of a program's solution without using any strict programming language syntax or underlying technology considerations.*

Objective Type Questions — OTQs

Multiple Choice Questions

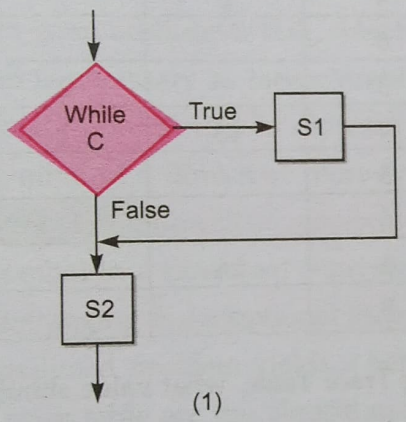
1. What is an algorithm ?
   (*a*) A set of steps to solve a problem
   (*b*) Software that analyses data
   (*c*) A hardware device that stores data
   (*d*) All of these
2. What is pseudo-code ?
   (*a*) A diagrammatic representation of a set of instructions
   (*b*) An algorithm written out in words
   (*c*) A very specific programming language used by all computers
   (*d*) All of these
3. Which of the following would assign user input to a variable?
   (*a*) INPUT = myVar
   (*b*) myVar ← INPUT
   (*c*) INPUT ← myVar
   (*d*) All of these
4. What shape represents a decision in a flowchart ?
   (*a*) A diamond
   (*b*) A rectangle
   (*c*) An oval
   (*d*) None of these

5. What is decomposition ?
   (a) Breaking code down once it has been run
   (b) Breaking a problem down into smaller, more manageable sections
   (c) Breaking a problem into subroutines (d) Breaking big data into small data

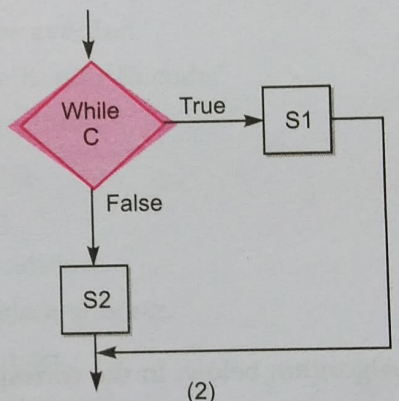
6. Which of the following are tools to design algorithms ?
   (a) Using variables and data (b) Using inputs and outputs
   (c) Using pseudo-code and flowcharts (d) Using functions and procedures

7. What is true for the term Decomposition ?
   (a) It uses computers to solve problems.
   (b) It represents 'real world' problems in a computer using variables and symbols and removes unnecessary elements from the problem.
   (c) It is the process of breaking down a large problem into smaller sub-problems.
   (d) It identifies the steps involved in solving a problem.

8. If C is true, S1 is executed in both the flowcharts, but S2 is executed in
   (a) (1) only (b) (2) only (c) both (1) and (2) (d) neither (1) nor (2)



(1)



(2)

9. Consider the following pseudocode and determine how many times the loop is iterated?

```
i := 0
while i ≠ 5
    i := i + 1
```

   (a) 4 (b) 5 (c) 6 (d) 0

10. If C is false just before the loop in the following code, the control flows through

```
1   S1
2   while C
3      S2
4   S3
```

   (a) S1 ; S3 (b) S1 ; S2 ; S3
   (c) S1 ; S2 ; S2 ; S3 (d) S1 ; S2 ; S2 ; S2 ; S3

11. Which of the following activities is algorithmic in nature ?
   (a) Assemble a bicycle. (b) Describe a bicycle.
   (c) Label the parts of a bicycle. (d) Explain how a bicycle works.

12. Which of the following activities is not algorithmic in nature ?
   (a) Multiply two numbers. (b) Draw a kolam.
   (c) Walk in the park. (d) Braid the hair.

13. The process of checking the correctness of an algorithm by checking the variables' values after every steps, without actually running the program is,
(a) No execution (b) Dry run (c) Dry output (d) Dry test

14. The purpose of a trace table is that
(a) it used to record the results from each step in an algorithm;
(b) it used to record the image from flowchart
(c) it used to store variable names
(d) it used to store the output

15. Look at the Algorithm below. In the corresponding Trace Table, what value should 'E' be ?

Algorithm

```
1   number = 3
2   PRINT number
3   FOR i from 1 to 3
4       number = number + 5
5       PRINT number
```

(a) 3
(b) 8
(c) 13
(d) 18

Trace Table

Line	number	i	OUTPUT
1	3		
2			3
3		1	
4	8		
5			8
3		2	
4	13		
5			13
3		E	
4	16		
5			F

16. Look at the Algorithm below. In the corresponding Trace Table, what value should 'F' be ?

Algorithm

```
1   number = 3
2   PRINT number
3   FOR i from 1 to 3
4       number = number + 5
5       PRINT number
```

(a) 3
(b) 8
(c) 13
(d) 18

Trace Table

Line	number	i	OUTPUT
1	3		
2			3
3		1	
4	8		
5			8
3		2	
4	13		
5			13
3		3	
4	18		
5			F

17. What are the common factors for comparing the two algorithms doing the same work ?
(a) Space (b) Time
(c) programming language (d) both (a) and (b) (e) (a), (b), and (c).

Fill in the Blanks

1. An _____ is a plan, a set of step-by-step instructions to solve a problem.
2. Using _____ we can break down the problem into smaller parts.
3. Writing in _____ is similar to writing in a programming language.
4. A _____ is a diagram that represents a set of instructions.
5. An _____ is simply a sequence of steps for completing a task.
6. Algorithms can be represented in many ways, the most commonly used being _____ and _____ .
7. Algorithms are compared through their _____ wise efficiency and _____ wise efficiency.
8. Testing an algorithm by viewing the impact of code on variables, using pen and paper is called _____ .
9. The table that stores the intermediate results of the code on a variable while doing dryrun, is called _____ table.
10. The _____ is similar to a code but is not based on any programming language.

True/False Questions

1. An algorithm is the step by step solution of a problem.
2. While solving a problem, you start coding from the first step.
3. Decomposition is an additional task, which may be avoided.
4. Decomposition is necessary as larger problems are harder to code.
5. A flowchart is a graphical tool to represent an algorithm.
6. A rectangle symbol in a flowchart represent process.
7. A diamond symbol in a flowchart represents a loop.
8. A diamond symbol in a flowchart represents a condition.
9. Testing and debugging is an optional step in problem solving.
10. A thorough analyzed problem yields a better solution.
11. Dry run and trace table are the algorithm verification tools.
12. Trace table always yields correct output.

NOTE : Answers for OTQs are given at the end of the book.

Solved Problems

1. *Mention the steps you would follow while writing a program.*

Solution. The steps to creating a working program are :

(*i*) Understand the problem well.

(*ii*) Analyze the problem to
- identify minimum number of inputs required for output.
- identify processing components.

(*iii*) Design the program by
- deciding step by step solution.
- breaking down solution into simple steps.

(*iv*) Code the program by
- identifying arithmetic and logical operations required for solutions.
- using appropriate control structures such as conditional or looping control structure.

(v) Test and Debug your program by
- finding errors in it.
- rectifying the errors.

(vi) Complete your documentation.

(vii) Maintain your program.

2. *What is an algorithm ?*

Solution. An Algorithm refers to a sequence of instructions, a finite set of commands or a method of working. It can be represented as a sequence of instructions to be carried out until an end point is reached as per the rules, conditions or sequence by which the computer or people tackle a problem or situation.

3. *Discuss two common tools for developing an algorithm.*

Solution. Two common tools for developing an algorithm are **flowchart** and **pseudocode**.

A **flowchart** is a diagrammatic or pictorial/graphical representation that illustrates the sequence of operations to be performed for the solution of a problem.

Pseudocode is an informal high-level description of a computer program or algorithm. It is written in symbolic code in English which must be translated into a proper code using a specific programming language.

4. *What do you understand by analysing an algorithm ?*

Solution. Problem analysis is an important phase as its success leads to the final solution. In problem analysis, mainly following things are carried out :
- identifying processing components.
- identifying the relationships among processing components.

5. *What is testing and debugging ?*

Solution. Testing is the process of finding errors in a program and debugging is the process of correcting errors found during the testing process. Thus, the testing and debugging involves :
- finding errors in the program.
- rectifying the errors, which have been found.

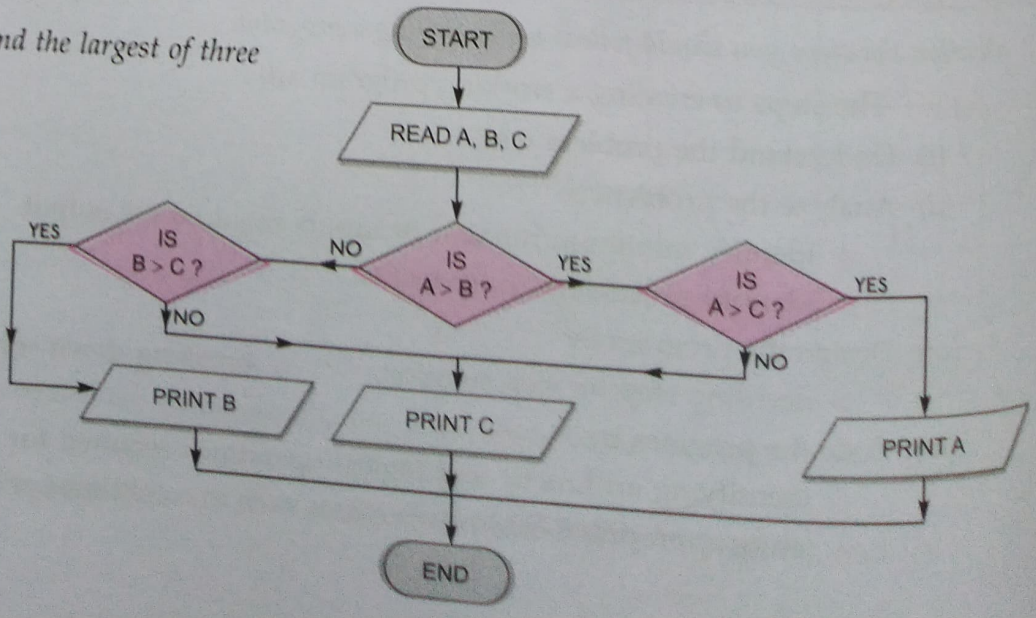
6. *How do you implement Decomposition while problem solving ?*

Solution. Decomposition is implemented by dividing the main algorithm into functions. Then one constructs each function independently of the main algorithm and other functions. Finally, the main algorithm is constructed using the functions.

When we use the functions, it is enough to know the specification of the function without knowing how the function is implemented.

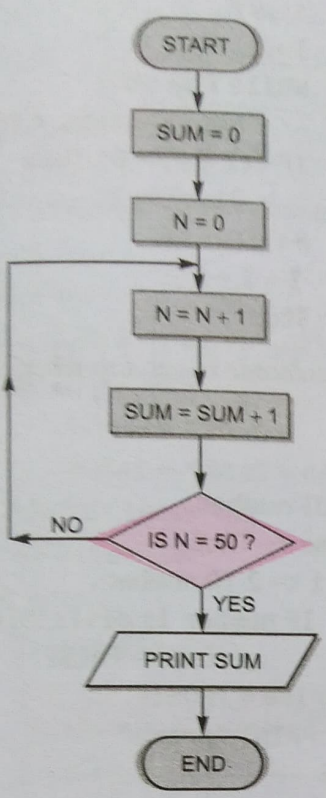
7. *Draw a flowchart to find the largest of three numbers A, B, and C.*

Solution.



8. *Draw a flowchart to find the sum of first 50 natural numbers.*

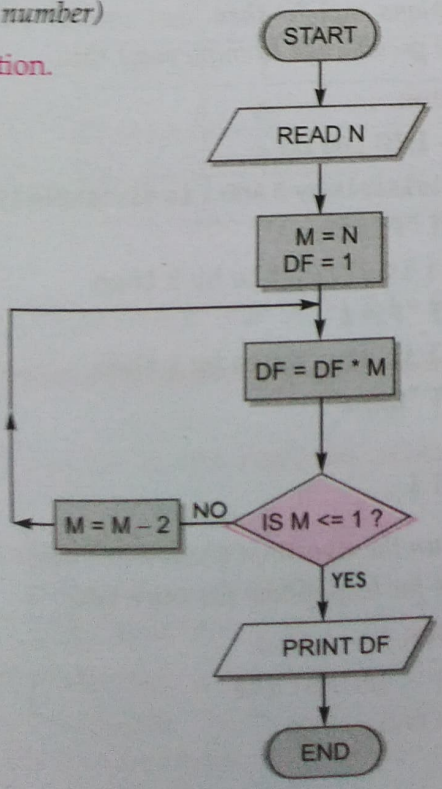
Solution.



9. *Draw a flowchart for computing double factorial of N (N!!), where* $N!! = N \times (N-2) \times (N-4) \times \ldots \times (N-k)$;

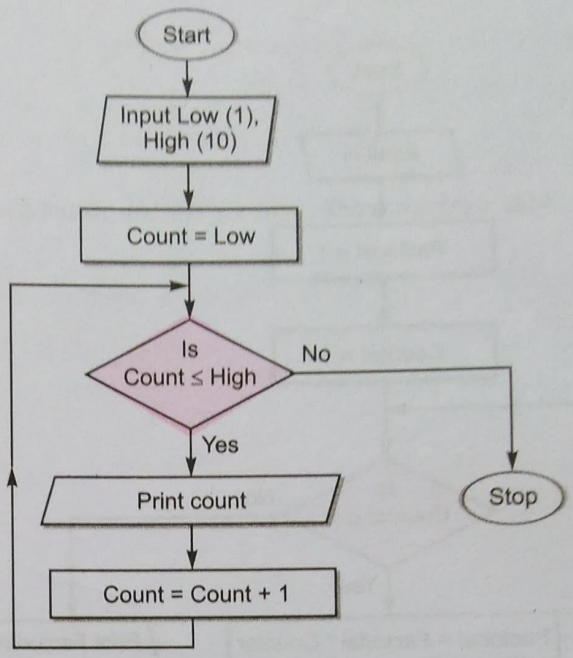
where k is multiple of 2 and $(N-k) >= 1$. *(N is an even number)*

Solution.



10. *Draw a flow chart to count and print from 1 to 10.*

Solution.



11. *Write pseudocode to count and print from 1 to 10.*

Solution.

```
1.  INPUT Low(1), High(10)
2.  Count = 1
3.  WHILE (Count <= High)
    Repeat steps 4 through 5
4.      PRINT Count
5.      Count = Count + 1
# end of while
```

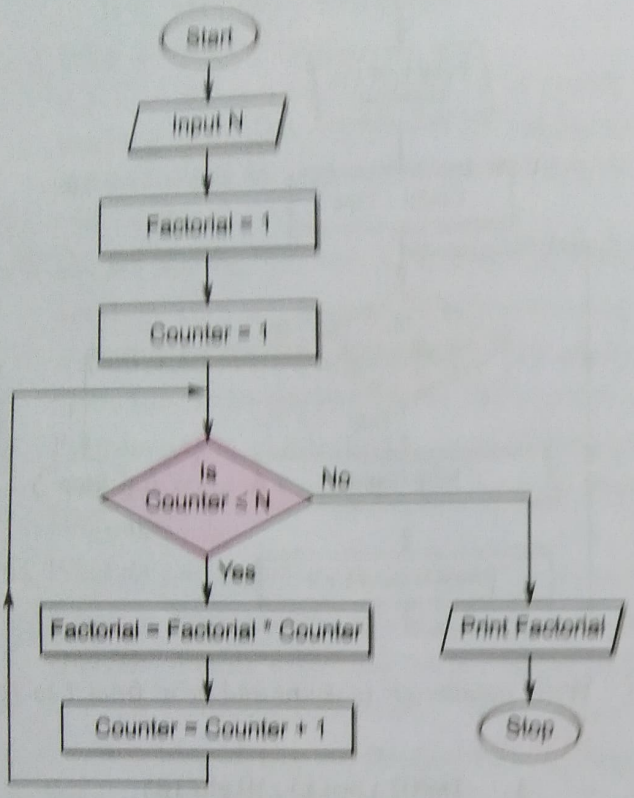
12. *Write pseudocode to calculate the factorial of a number (N).*

Solution.

```
1: Input N
2: Factor = 1
3: Counter = 1
4: While (Counter £ N)
       Repeat steps 4 through 6
5:   Factor = Factor * Counter
6:   Counter = Counter + 1
7: Print (N, Factor)
```

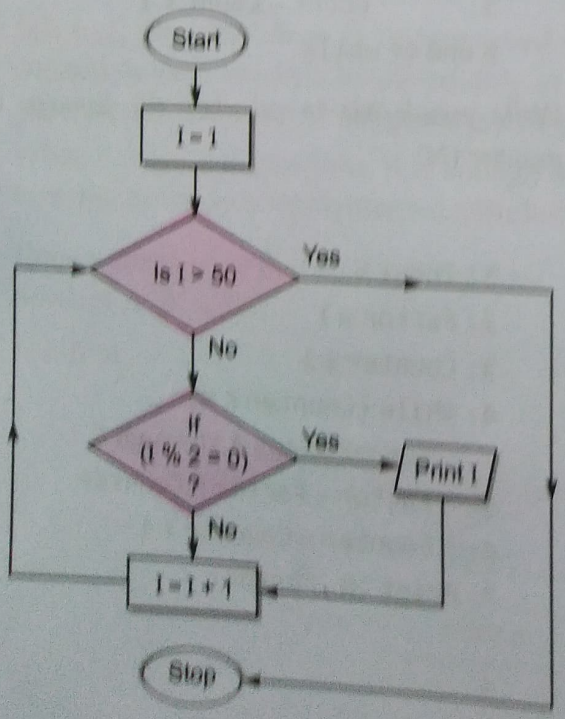
13. Draw a flow chart to calculate the factorial of a number (N).

Solution:



14. Draw a flowchart to print Even numbers between 1 to 50.

Solution:



15. Write pseudocode to print Even numbers between 1 to 50.

Solution:

```
1   Start
2   I = 1
3   While I <= 50
        Repeat steps 4 through 5
4   IF ((I % 2) = 0) THEN
        Display I
    # END of IF
5   I = I + 1
7   Stop
```

16. Write pseudocode to get a number and check if it is prime.

Solution:

```
INPUT number
prime <- TRUE
FOR i <- 2 TO number
    IF number is divisible by i then
        prime <- FALSE
IF prime = TRUE:
    PRINT "prime"
ELSE
    PRINT "not prime"
```

17. For every number from 1 to 100, output Fizz if the number is divisible by 3, output Buzz if the number is divisible by 5, and output FizzBuzz if the number is divisible by both 3 and 5. If none of these conditions match, then just output the number. Write pseudocode to implement this.

Solution:

```
FOR i <- 1 TO 100 DO
IF i is divisible by 3 AND i is divisible by 5 then
    PRINT "FizzBuzz"
ELSE IF i is divisible by 3 then
    PRINT "Fizz"
ELSE IF i is divisible by 5 then
    PRINT "Buzz"
ELSE
    PRINT i
```

18. Dry run the algorithm given below which contains a simple for loop. Show the trace table.

```
FOR i <- 1 TO 4
    OUTPUT i*2
# END OF FOR
```

Solution. **Trace table**

i	Output
1	2
2	4
3	6
4	8

19. *Dry run the algorithm given below which contains a for loop inside another for loop. Show the trace table.*

```
num ← 0
total ← 0
FOR i ← 1 TO 3
    FOR j ← 1 TO 3
        num ← j * i
        total ← total + num
    # END of inner FOR
# END of outer FOR
PRINT total
```

Solution. **Trace table**

i	*j*	*num = j * i*	*total = total + num*	*OUTPUT*
		0	0	
1	1	1	1	
	2	2	3	
	3	3	6	
2	1	2	8	
	2	4	12	
	3	6	18	
3	1	3	21	
	2	6	27	
	3	9	36	
				36

Guidelines to NCERT Questions [NCERT Chapter 4]

1. *Write pseudocode that reads two numbers and divides one by another and displays the quotient.*

Ans.

```
INPUT num1, num2
quotient ← num1/num2
PRINT quotient
```

2. *Two friends decide who gets the last slice of a cake by flipping a coin five times. The first person to win three flips wins the cake. An input of 1 means player 1 wins a flip, and a 2 means player 2 wins a flip. Design an algorithm to determine who takes the cake ?*

Ans.
```
INPUT num
IF num = 1 THEN
    PRINT 'Player1 has won'
ELSE IF num = 2 THEN
    PRINT 'Player2 has won'
ELSE
    PRINT 'Enter 1 or 2 only'
```

3. *Write the pseudocode to print all multiples of 5 between 10 and 25 (including both 10 and 25).*

Ans.
```
FOR i in sequence 10..25
    IF i modulus 5 = 0 THEN
        PRINT i
    #End of IF
#End of FOR
```

4. *Give an example of a loop that is to be executed a certain number of times.*

Ans. The FOR loop, can be executed a certain number of times *e.g.*,
```
For i in sequence 1..5
    PRINT i
```

5. *Suppose you are collecting money for something. You need ₹200 in all. you ask you parents, uncles and aunts as well as grandparents. Different people may give either ₹10, ₹10 or even ₹50. You will collect till the total becomes 200. Write the algorithm.*

Ans.
```
Total = 0
WHILE Total < 200
    INPUT money
    Total = Total + Money
#End of while
```

6. *Write the pseudocode to print the bill depending upon the price and quantity of an item. Also print Bill GST, which is the bill after adding 5% of tax in the total bill.*

Ans.
```
INPUT itemname, price, qty
billamount = price * qty
gstamount = billamount * 0.05
PRINT itemname, price, billamount
PRINT "GST = ", gstamount
PRINT 'Total amount = ', billamount + gstamount
```

7. *Write pseudocode that will perform the following :*

(a) *Read the marks of three subjects : Computer Science, Mathematics and Physics, out of 100.*

(b) *Calculate the aggregate marks.*

(c) *Calculate the percentage of marks.*

Ans.
```
INPUT cmarks, mmarks, pmarks
total = cmarks + mmarks + pmarks
perc = (total/300) * 100
PRINT cmarks, mmarks, pmarks
PRINT 'Aggregate = ', total
PRINT 'Percentage = ', perc
```

8. *Write an algorithm to find the greatest among two different numbers entered by the user.*

Ans.
```
INPUT num1, num2
IF num1 > num2
    PRINT num1
ELSE IF num2 > num1
    PRINT num2
ELSE
    PRINT 'both are equal'
```

9. *Write an algorithm that performs the following : Ask a user to enter a number. If the number is between 5 and 15, write the word GREEN. If the number is between 15 and 25, write the word BLUE. If the number is between 25 and 35, write the word ORANGE. If it is any other number, write that ALL COLOURS ARE BEAUTIFUL*

Ans.
```
INPUT num
IF num >= 5 and <= 15 THEN
    PRINT 'GREEN'
ELSE IF num >= 15 and <= 25 THEN
    PRINT 'BLUE'
ELSE
    PRINT 'ALL COLOURS ARE BEAUTIFUL'
```

10. *Write an algorithm that accepts four numbers as input and find the largest and smallest of them.*

Ans.
```
INPUT n1, n2, n3, n4
IF n1 > n2 and n1 > n3 and n1 > n4 THEN
    PRINT n1 'is greatest'
ELSE IF n2 > n1 and n2 > n3 and n2 > n4 THEN
    PRINT n2 'is greatest'
ELSE IF n3 > n1 and n3 > n2 and n3 > n4 THEN
    PRINT n3 'is greatest'
ELSE IF n4 > n1 and n4 > n2 and n4 > n3 THEN
    PRINT n4 'is greatest'.
IF n1 < n2 and n1 < n3 and n1 < n4 THEN
    PRINT n1 'is smallest'
```

```
ELSE IF n2 < n1 and n2 < n3 and n2 < n4 THEN
    PRINT n2 'is smallest'
ELSE IF n3 < n1 and n3 < n2 and n3 < n4 THEN
    PRINT n3 'is smallest'
ELSE IF n4 < n1 and n4 < n2 and n4 < n3 THEN
    PRINT n4 'is smallest'.
```

11. *Write an algorithm to display the total water bill charges of the month depending upon the number of units consumed by the customer as per the following criteria :*

- for the first 100 units @ 5 per unit
- for next 150 units @ 10 per unit
- more than 250 @ 20 per unit

Ans.

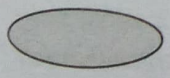
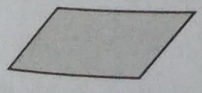
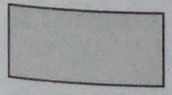
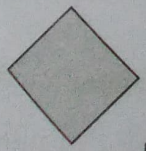
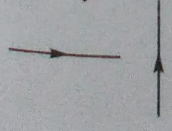
```
INPUT units
IF units > 250 THEN
   bill = (units-250) * 75 + 150 * 10 + 100 * 5
ELSE IF units > 100 THEN
   bill = (units-100) * 10 + 100 * 5
ELSE
   bill = units * 5
PRINT units, bill
```

12. *What are conditionals ? When they are required in a program ?*

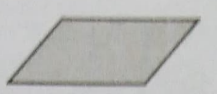
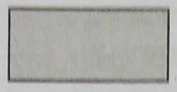
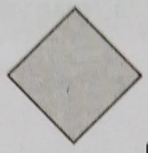
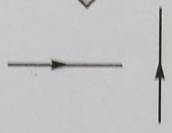
Ans. Conditionals are the statements that test a condition.

Conditionals are required in a program where the control has to take different flow-of-action depending upon a condition's result.

13. *Match the pairs*

Flowchart Symbols	Functions
	Flow of Control
	Process Step
	Start/Stop of the Process
	Data
	Decision Making

Ans.

Flowchart Symbols	Functions
	Start/Stop of the Process
	Data
	Process Step
	Decision Making
	Flow of Control

14. *Following is an algorithm for going to school or college. Can you suggest improvements in this to include other options ? Reach_School_Algorithm*

(*a*) Wake up (*b*) Get ready
(*c*) Take lunch box (*d*) Take bus
(*e*) Get off the bus
(*f*) Reach school or college

Ans. It can be modified to incorporate situations where a different course of action is required, *e.g.*, if bus is missed.

```
Wake up
Get Ready
Take Lunch Box
IF taken bus THEN
    Ride the bus
    Get off the bus
ELSE    #if bus missed
    Take other means of transport
    Get down at school
Reach School or College.
```

15. *Write a pseudocode to calculate the factorial of a number (Hint. Factorial of 5, written as 5! = 5×4×3×2×1).*

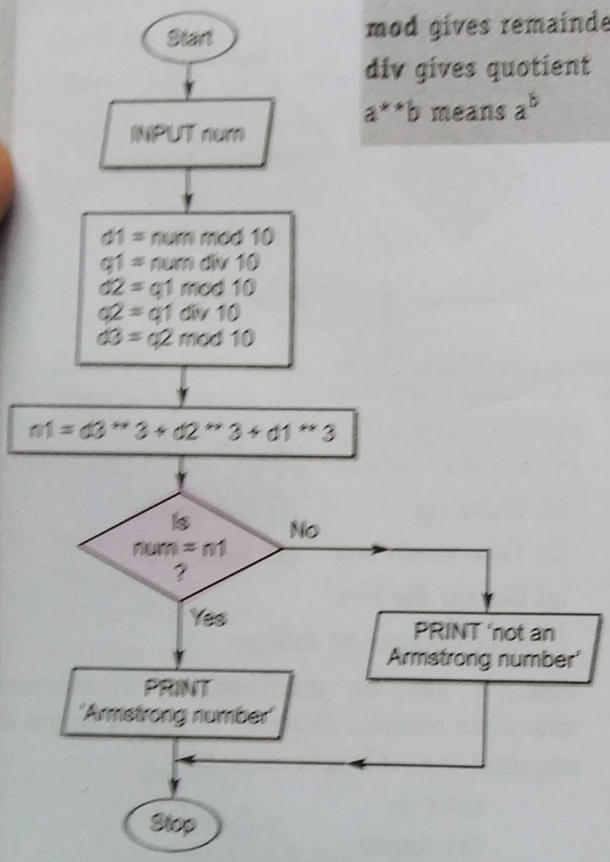
Ans.

```
INPUT num
F = 1
WHILE num > 1
    F = F * num
    num = num - 1
#end of while
PRINT F
```

16. *Draw a flowchart to check whether a given number is an Armstrong number. An Armstrong number of three digits is an integer such that the sum of the cubes of its digits is equal to the number itself. For example, 371 is an Armstrong number since 3**3+17**3+1**3=371.*

Ans.

NOTE
mod gives remainder
div gives quotient
a**b means a^b



17. *Following is an algorithm to classify numbers as "Single Digit", "Double Digit" or "Big".*

Classify_Numbers_Algo

```
INPUT Number
If Number < 9
    "Single Digit"
Else If Number < 99
    "Double Digit"
Else
    "Big"
```

Verify for (5, 9, 47, 99, 100, 200) and correct the algorithm if required.

Ans.

```
1. INPUT Number
2. IF Number < 9
3.     "Single Digit"
4. Else IF Number < 99
5.     "Double Digit"
6. ELSE
7.     "Big"
```

Number	Lines Executed	OUTPUT	
5	1, 2, 3	Single Digit	✓
9	1, 2, 4, 5	Double Digit	×
47	1, 2, 4, 5	Double Digit	✓
99	1, 2, 4, 6, 7	Big	×
100	1, 2, 4, 6, 7	Big	✓
200	1, 2, 4, 6, 7	Big	✓

The error in given algorithm is that it should also test for equality *i.e.*, as :

```
INPUT Number
IF Number <= 9
    "Single Digit"
ELSE IF Number <= 99
    "Double Digit"
ELSE
    "Big"
```

18. *For some calculations, we want an algorithm that accepts only positive integers upto 100.*

Accept_1to100_Algo

```
INPUT Number
IF(0 <= Number) AND (Number <= 100)
    ACCEPT
Else
    REJECT
```

(*a*) *On what values will this algorithm fail ?*

(*b*) *Can you improve the algorithm ?*

Ans.

(*a*) The given algorithm will fail at input number as zero (0).

(*b*) Corrected algorithm :

```
INPUT Number
IF (0 < Number) AND (Number <= 100)
    ACCEPT
ELSE
    REJECT
```

GLOSSARY

Flow Chart	Graphical representation of a problem's solution.
Algorithm	Step wise logic of a program in normal English.
Pseudocode	Logic written in English which looks like a programming language but not actually a programming language.
Dry Run	Testing a program with pen and paper, by tracing the impact of each statement on variables.

Assignments

Type A : Short Answer Questions/Conceptual Questions

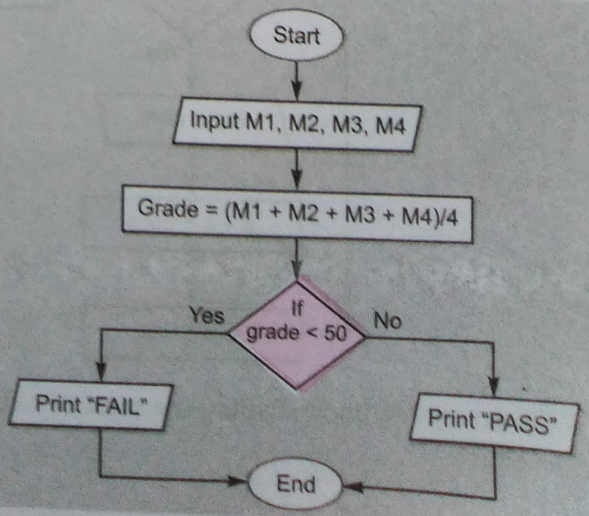
1. What are the phases of program solving cycle ?
2. What do you do while analysing a problem ?
3. What is done during coding phase ?
4. What is testing and debugging ?
5. Distinguish between a condition and a statement.
6. Draw a flowchart for conditional statement.
7. Both conditional statement and iterative statement have a condition and a statement. How do they differ ?
8. What is decomposition ?
9. Name some tools used for problem solution development.
10. What is the difference between an algorithm and a program ?
11. What is dry run ? How is it useful ?
12. What is trace table ?

Type B : Application Based Question

1. Write an algorithm to find the square of a number.
2. Draw a flowchart to solve the problem of a non-functioning light bulb.
3. Draw a flowchart for calculating grade from marks percentage.
4. Write an algorithm to double a number in two different ways : (*i*) $n+n$, (*ii*) $2 \times n$.
5. Write an algorithm and draw a flowchart to determine if a student passed the exam or not.(Note there are 4 subject papers and passing average is 50 or more.)

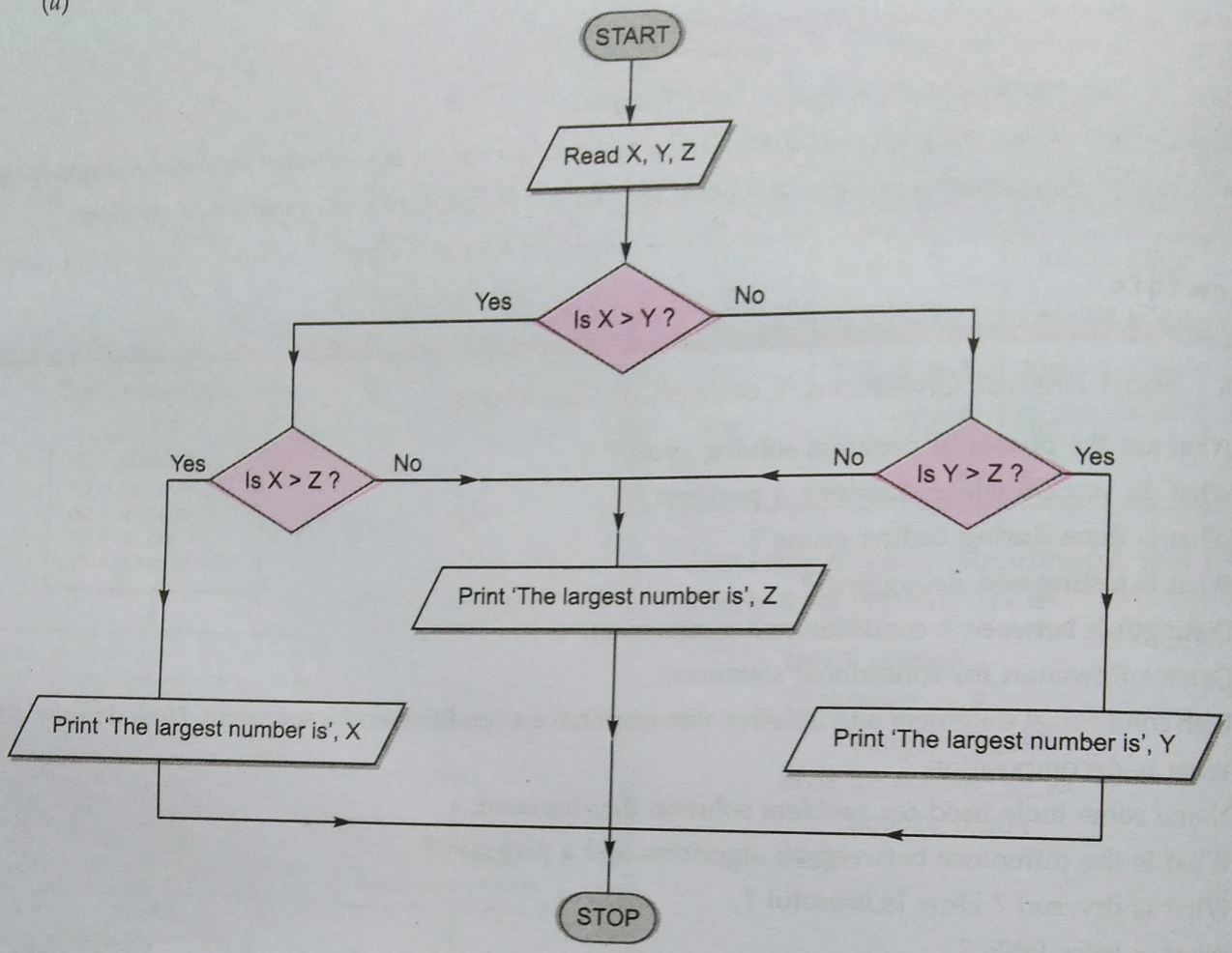
[Hint :

```
Step 1:  Input M1,M2,M3,M4
Step 2:  GRADE ß (M1 + M2 + M3 + M4)/4
Step 3:  if (GRADE < 50) then
             Print "FAIL"
         else
             Print "PASS"
         # end of if
```

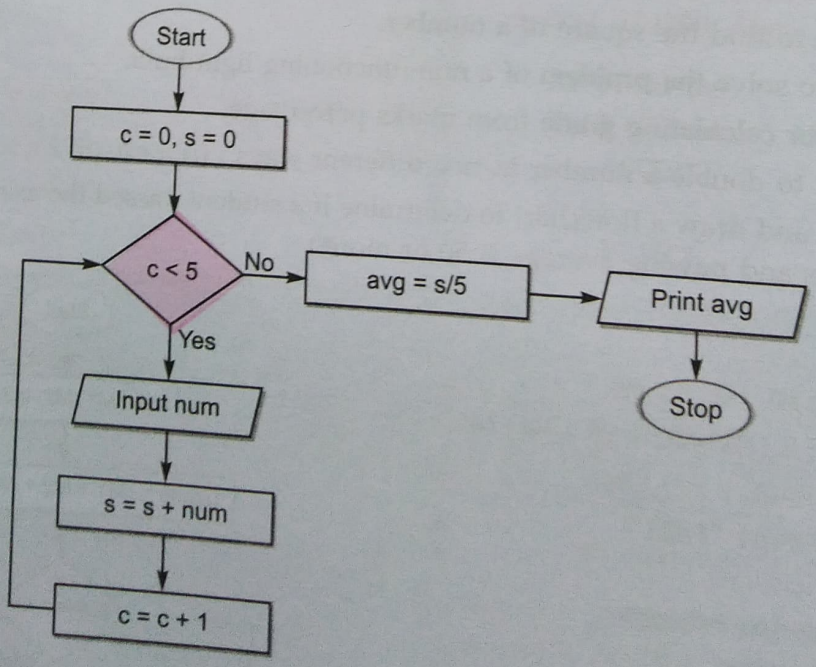


6. Write pseudocode for the following :

(*a*)



(*b*)



Write an algorithm to display the sum of two numbers entered by user, using both pseudocode and flowchart.

*Draw a **flowchart**, write an **algorithm** and **pseudo code** for the following questions 8-24*

8. To find the area and perimeter of a rectangle.
9. To calculate the area and the circumference of a circle.
10. To calculate the simple interest .
11. To check whether a year is a leap year or not.
12. To check if a number is a positive or negative number.
13. To check if a number is an odd or even number.
14. To categorise a person as either child (< 13), teenager (>= 13 but < 20) or adult (>= 20), based on age specified.
15. To print all natural numbers up to *n*.
16. To print n odd numbers.
17. To print square of a number.
18. To accept 5 numbers and find their average.
19. To accept numbers till the user enters 0 and then find their average.
20. To print squares of first n numbers.
21. To print the cube of a number.
22. To print to print cubes of first n numbers.
23. To find sum of n given numbers.
24. To find factorial of a given number.

25. Given the following pseudo code :

```
Use variables sum, product, number1, number2 of type real
display "Input two numbers"
accept number1, number2
sum = number1 + number2
print "The sum is", sum
product = number1 * number2
print "The Product is ", product
end program
```

Draw a flow chart for the same and dry run the given pseudocode if it is working fine.

26. Given the following pseudo code :

```
Use variables: choice, of the type character
ans, number1, number2, of type integer
display "choose one of the following"
display "m for multiply"
display "a for add"
display "s for subtract"
accept choice
display "input two numbers you want to use"
accept number1, number2
if choice = m then ans = number1 * number2
if choice = a then ans = number1 + number2
if choice = s then ans = number1 - number2
display ans
```

Draw a flow chart for the same and dry run the given pseudocode if it is working fine.

27. Given the following pseudo code :

```
Use variables: mark of type integer
 If mark >= 80, display "distinction"
 If mark >= 60 and mark < 80, display "merit"
 If mark >= 40 and mark < 60, display "pass"
 If mark < 40 display, "fail"
```

Draw a flow chart for the same and dry run the given pseudocode if it is working fine.

28. Given the following pseudo code:

```
Use variables: category of type character
Display "input category"
Accept category
If category = 'U'
  Display "insurance is not available"
Else If category = 'A' then
  Display "insurance is double"
Else If category = 'B' then
  Display "insurance is normal"
Else  If category = 'M' then
  Display "insurance is medically dependent"
Else
  Display "entry invalid"
```

Draw a flow chart for the same and dry run the given pseudocode if it is working fine.

29. Given the following pseudo code :

```
Use variable: number of type real
DISPLAY "Type in a number or zero to stop"
ACCEPT number
WHILE number  0
  Square = number * number
  DISPLAY "The square of the number is", square
  DISPLAY "Type in a number or zero to stop"
  ACCEPT number
ENDWHILE
```

Draw a flow chart for the same and dry run the given pseudocode if it is working fine.