

Chapter 4

Introduction to Problem Solving

Problem solving

Problem solving is a process of identifying the problem, developing an algorithm for the problem and implementing the algorithm to create a computer program.

4.1 Steps for Problem Solving

Problem solving begins with the precise identification of the problem and ends with a complete working solution in terms of a program or software.

1. Analysing the problem
2. Developing an Algorithm
3. Coding
4. Testing and Debugging

1. Analysing the problem

It is important to clearly understand a problem before we begin to find the solution for it. By analysing a problem, we would be able to figure out what are the inputs that our program should accept and the outputs that it should produce.

2. Developing an Algorithm

An **algorithm** is a **finite sequence of well-defined, step-by-step instructions** designed to solve a specific problem or perform a task.

3. Coding

Coding is the process of **translating an algorithm into a programming language** so that a computer can understand and execute it.

4. Testing and Debugging

- Testing is the process of **checking a program against various parameters** to ensure it meets user requirements, produces correct outputs, and responds within expected time.
- Debugging is the process of **identifying, rectifying, and removing errors or defects** found during testing so that the program works correctly.

4.2 Algorithm

An **algorithm** is a **finite sequence of well-defined, step-by-step instructions** designed to solve a specific problem or perform a task.

Advantage:

Increase the reliability, accuracy and efficiency of obtaining solutions.

Characteristics of a good algorithm

- Precision — the steps are precisely stated or defined.
- Uniqueness — results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
- Finiteness — the algorithm always stops after a finite number of steps.
- Input — the algorithm receives some input.
- Output — the algorithm produces some output.

4.3 Representation of Algorithms

There are two common methods of representing an algorithm —

1. Flowchart
2. Pseudocode

4.3.1 Flowchart — Visual Representation of Algorithms

A flowchart is a visual representation of an algorithm. A flowchart is a diagram made up of boxes, diamonds and other shapes, connected by arrows.

There are standardised symbols to draw flowcharts. Some are given in Table 4.1.

Table 4.1 Shapes or symbols to draw flow charts

Flowchart symbol	Function	Description
	Start/End	Also called "Terminator" symbol. It indicates where the flow starts and ends.
	Process	Also called "Action Symbol," it represents a process, action, or a single step.
	Decision	A decision or branching point, usually a yes/no or true/false question is asked, and based on the answer, the path gets split into two branches.
	Input/Output	Also called data symbol, this parallelogram shape is used to input or output data
	Arrow	Connector to show order of flow between shapes.

Example 4.1: Write an algorithm to find the square of a number.

Step1: Start

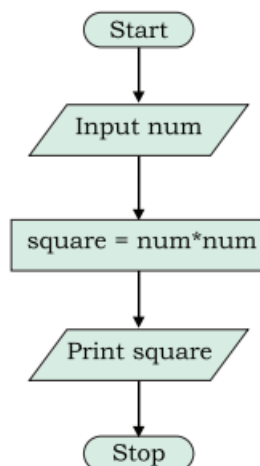
Step 2: Input number

Step 3: Square = number*number

Step 4: Print Square

Step 5: Stop

Flowchart to calculate square of a number



4.3.2 Pseudocode

It is a non-formal language that helps programmers to write algorithm.

“Pseudo” means “not real,”

“Pseudocode” means “not real code”.

Keywords while writing pseudocode: INPUT, COMPUTE, PRINT, INCREMENT, DECREMENT, IF/ELSE, WHILE, TRUE/FALSE

Example 1: Write an algorithm to display the sum of two numbers entered by user, using both pseudocode and flowchart.

Pseudocode for the sum of two numbers will be:

Step 1: input num1

Step 2: input num2

Step 3: COMPUTE Result = num1 + num2

Step 4: PRINT Result

The **flowchart** for this algorithms is given in Figure 1.

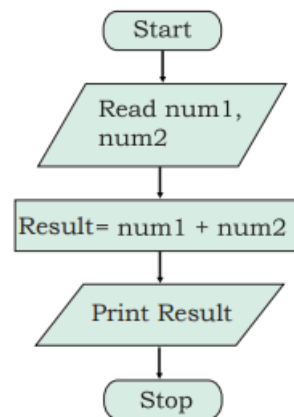


Figure 1: Flowchart to display sum of two numbers

Example 2: Write an algorithm to calculate area and perimeter of a rectangle, using both pseudocode and flowchart.

Pseudocode for calculating area and perimeter of a rectangle.

Step 1: input length

Step 2: input breadth

Step 3: compute $\text{Area} = \text{length} * \text{breadth}$

Step 4: print Area

Step 5: compute $\text{Perim} = 2 * (\text{length} + \text{breadth})$

Step 6: print Perim

The **flowchart** for this algorithm is given in **Figure 2**.

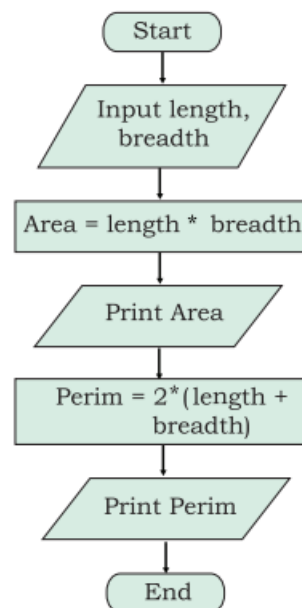


Figure 2: Flowchart to calculate area and perimeter of a rectangle

Benefits of Pseudocode

- Helps programmers plan the logic step-by-step in a **human-readable format**.
- Prevents Missing Steps
- Easy to Understand

4.4 Flow of Control

It explains movement of data between the statements in an algorithm.

Types of flow of control

1. Sequence
2. Selection
3. Repetition

1. Sequence

The statements are executed one after another in a sequence.

2. Selection

There are some cases where we need to select one option out of multiple options in such cases we use selection statements.

Example: If else.

Example 1: Let us write an algorithm to check whether a number is odd or even.

Pseudocode of the algorithm can be written as follows:

1. PRINT "Enter the Number"
2. INPUT number
3. IF number MOD 2 == 0 THEN
4. PRINT "Number is Even"
5. ELSE
6. PRINT "Number is Odd"

The flowchart representation of the algorithm is shown in Figure 3.

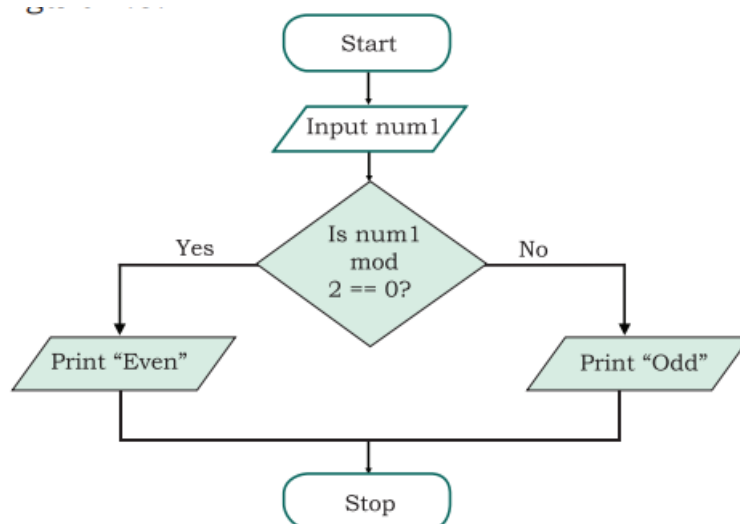


Figure 3: Flowchart to check whether a number is even or odd

Example 2: Let us write a pseudocode and draw a flowchart where multiple conditions are checked to categorise a person as either child (≤ 13 but ≥ 20), based on age specified:

Pseudocode is as follows:

1. INPUT Age
2. if Age < 13 then
3. PRINT "Child"
4. else
5. if Age < 20 then
6. PRINT "Teenager"
7. else
8. PRINT "Adult"

The flowchart representation of the algorithm is shown in Figure 4

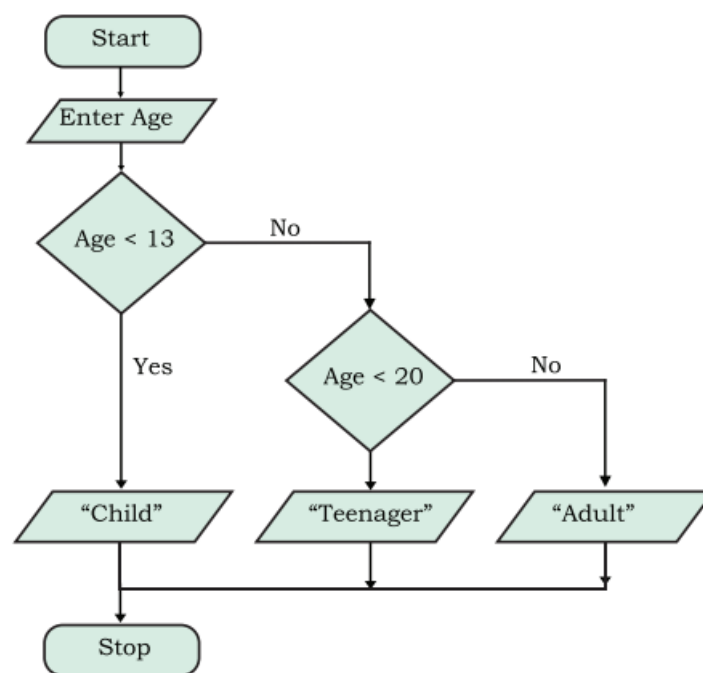


Figure 4: Flowchart to check multiple conditions

3. Repetition

Set of statements are repeatedly executed until some condition is satisfied is called looping/repetition/Iteration.

Example 1: Write pseudocode and draw a flowchart to accept 5 numbers and find their average.

Pseudocode will be as follows:

Step 1: Set count = 0, sum = 0

Step 2: While count<5, REPEAT STEPS 3 to 5

Step3: INPUT a number to num

Step 4: sum = sum+num

Step 5: count = count + 1

Step 6: COMPUTE average = sum/5

Step 7: Print average

The flowchart representation is shown in Figure 5.

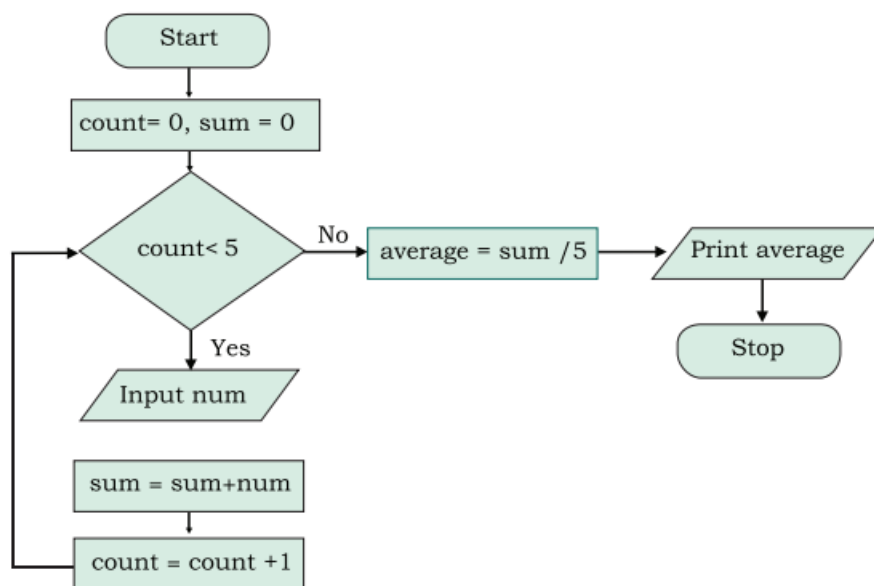


Figure 5: Flowchart to Calculate the Average of 5 Numbers

Example 2: Write pseudocode and draw flowchart to accept numbers till the user enters 0 and then find their average.

Pseudocode is as follows:

Step 1: Set count = 0, sum = 0

Step 2: Input num

Step 3: While num is not equal to 0, repeat Steps 4 to 6

Step 4: sum = sum + num

Step 5: count = count + 1

Step 6: Input num

Step 7: Compute average = sum/count

Step 8: Print average

The flowchart representation is shown in Figure 6.

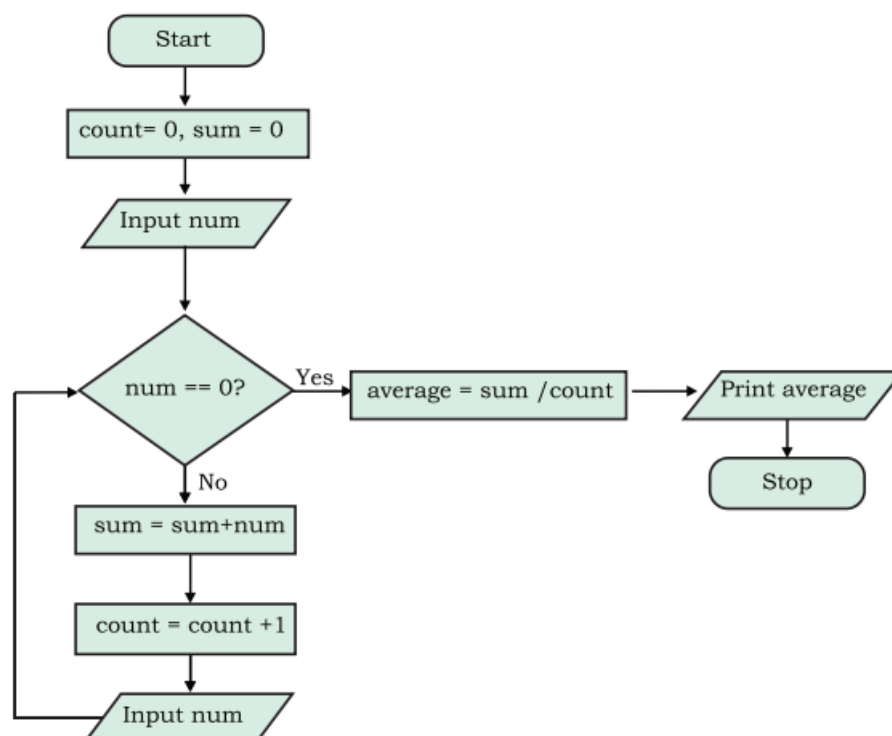


Figure 6: Flowchart to accept numbers till the user enters 0

4.5 Verifying Algorithms

The algorithm written must be verify in every possible way before writing in any programming language.

Dry Run: The method of taking an input and running through the steps of algorithm is called Dry Run.

Advantages:

1. Identify incorrect steps.
2. Spot missing details or specifics.
3. Modify or improve algorithm if needed.

4.6 Comparison of Algorithm

They can be more than one approach to solve a problem choosing the best method by comparing with different approaches is done by.

1. Time complexity:

It is time taken by the algorithm to complete its execution is called Time complexity.

2. Space Complexity:

Amount of memory space required by the algorithm to complete its run is called Space complexity.

- The best algorithm is selected by analysing different approaches and choosing the one that gives correct results with minimum time complexity and space complexity for the given problem.

4.7 Coding

Once an algorithm is finalized, it is written in a **high-level programming language (source code)**.

The process of converting an algorithm into computer executable programme by using syntax of a programming language is called coding.

Programs are developed not only to work on a computer, mobile or a web browser, but it may also be written for embedded systems like digital watch, mp3 players, traffic signals or vehicles, medical equipment's and other smart devices.

4.8 Decomposition

Decomposition means breaking down a **complex problem** into **simpler sub-problems**. Each sub-problem is solved individually, then combined to form the complete solution is called Decomposition.

Example

- **Railway Reservation System:**
 - Complex task → divided into components (booking, cancellation, seat allocation, payment, etc.).
 - Each component designed separately, then integrated.

Advantages

- Simplifies problem-solving.
- Each sub-problem can be studied in detail.
- Different teams can work independently on different parts.
- Experts can handle specific sub-problems.
- Encourages modularity and reusability.

• Applications in Real Life

- Mathematics and science problem solving.
- Event management in schools.
- Weather forecasting.
- Delivery management systems.
- Many other complex systems.

• Final Step

- After solving sub-problems, test them for correctness.
- Integrate them to obtain the complete solution.

Algorithms:

1. **Write pseudocode that reads two numbers and divide one by another and display the quotient.**
 1. Input first number to variable A
 2. Input second number to variable B
 3. Compute $\text{Quotient} = A/B$
 4. Print Quotient
2. **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**

1. Start
2. Input C, M, P
3. Compute $\text{Sum} = C + M + P$
4. Compute $\text{Percentage} = (\text{Sum}/300) * 100$
5. Print Sum, Percentage

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

Input a number to variable N

1. Start
2. Set $f=1, i=1$
3. While $i \leq N$ do
4. Compute $f=f*i$
5. $i = i+1$
6. print fact

4. **Write pseudocode to classify the number as Single digit, Double digit or Big**

1. Start
2. Input number
3. If($\text{number} < 9$)
4. Print "Single digit"
5. Else
6. If($\text{number} < 99$)
7. Print "Double Digit"
8. Else
9. Print "Big"