5.3.2 Pseudocode

Pseudocode is an informal language that helps programmers describe steps of a program solution without using any programming language syntax. Pseudocode is a "text-based" algorithmic design tool. A pseudocode creates an outline or a rough draft of a program and 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 instructions are written in uppercase, variables in lowercase, 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, 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 updated as expected.

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

DRY RUN

A dry run is the process of programmer manually through their code to trace value of variables. There software involved in this

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 code is seen on the values of variables of the algorithm. As per the line of the code 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 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., a 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 the 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.*