**UNIT 1**

**INTRODUCTION TO COMPUTER PROBLEM SOLVING**

***Algorithms, building blocks of algorithms (statements, control flow, functions), notation (pseudo code, flow chart), algorithmic problem solving for socio economic conditions in global perspectives, simple strategies for developing algorithms (iteration, recursion), the efficiency of algorithms.***

**1.1 Algorithms**

An algorithm is a step by step procedure of solving a problem. The word “algorithm” is derived from the name of the 9th century Persian mathematician Al-Khwarizmi.

An algorithm is defined as a collection of unambiguous instructions which are executed in a specific sequence to produce an output in a finite amount of time for a given set of input data.

An algorithm can be written in English or in any standard representation.

An algorithm must possess the following properties

1. Finiteness: An algorithm must be executed finite number of times.

2. Definiteness: Each step of the algorithm must be accurate and clear.

3. Effectiveness: Each step must be effective, in the sense that it should be primitive (easily convertible into a program statement) and can be performed exactly in a finite amount of time.

4. Independent: The algorithm must be independent of any programming code.

5. Input/output: Each algorithm must take zero or more quantities as input data and give out one or more output values.

Methods for Developing an Algorithm

* List the data needed to solve the problem (input) and know what is the end result (output).
* Describe the various step to process the input to get the desired output. Break down the complex processes into simpler statements.
* Finally test the algorithm with different data sets.

Example: Algorithm for Addition of two numbers:

Step1: Start

Step 2: Get two numbers a and b as input

Step 3: Add the numbers a & b and store the result in c

Step 4: Print c

Step 5: Stop.

The above algorithm is to add two numbers a and b. The numbers are the input provided by the user .After adding the result is stored in a variable c and it is printed.

**1.2 Building Blocks of Algorithm:** The basic building blocks of an algorithm are Statements, Control flow and Functions.

**1.2.1 Statements:** Statement may be a single action in a computer.

In a computer statements might include some of the following actions

* input data-information given to the program
* process data-perform operation on a given input
* output data-processed result

**1.2.2 Control flow:** The process of executing the statements in the given sequence is called as control flow.

The three ways in executing the control flow are

* Sequence
* Selection
* Iteration

***Sequence***

The given set of instructions executed consecutively is called sequence execution.

Example: Algorithm to find the average of three numbers, the algorithm is as follows

Step1: Start

Step2: Get three numbers a, b, c as input

Step3: Find the sum of a, b and c

Step4: Divide the sum by 3

Step5: Store the result in variable d

Step6: Print d

Step7: Stop

The above algorithm finds the average of three numbers a, b and c. The numbers are the input provided by the user .After adding the total is divided by 3 and the result is stored in a variable d and it is printed.

***Selection***

A selection statement is transferring the program control to a specific part of the program based upon the condition.

If the condition is true, one part of the program will be executed, otherwise the other part of the program will be executed.

Example: Algorithm to find the Largest of two numbers, the algorithm is as follows

Step1: Start

Step 2: Get two numbers a and b as input

Step 3: IF a>b THEN

Step 4: PRINT “A is Large”

Step 5: ELSE PRINT “B is Large”

Step 6: ENDIF

Step 7: Stop

The above algorithm is used to find the Largest of two numbers a and b. The numbers are the input provided by the user .The number a and b are compared, If a is larger Print “A is Large” or if b is larger print “B is Large”.

***Iteration***

Iteration is a type of execution where a certain set of statements are executed again and again based upon condition. This type of execution is also called as looping or repetition.

Example: Algorithm to print all natural numbers up to n, the algorithm is as follows

Step 1: Start

Step 2: Get the value of n.

Step 3: Initialize i=1

Step 4: if (i<=n) go to step 5 else go to step 7

Step 5: Print i value and increment i value by 1

Step 6: Go to step 4

Step 7: Stop

The above algorithm is for printing first n natural numbers .The user provides the input. The first value, i is initialized. A loop is initialized. The first value is printed and the number is incremented. The i value is checked with n, the user input. The numbers are printed until i becomes greater than n.

**1.2.3 Functions**

Function is a sub program which consists of block of instructions that performs a particular task. For complex problems, the problem is been divided into smaller and simpler tasks during algorithm design.

Benefits of Using Functions

* Reduction in line of code
* code reuse
* Better readability
* Information hiding
* Easy to debug and test
* Improved maintainability

Example: Algorithm for addition of two numbers using function Main function ()

Step 1: Start

Step 2: Call the function add ()

Step 3: Stop

The above algorithm is to call the function add. This function is called as Main function or calling function.

**Subfunction add ()**

Step 1: Function start

Step 2: Get two numbers as input and store it in to a and b

Step 3: Add the numbers a & b and store the result in c

Step 4: Print c

Step 5: Return.

The above algorithm is called as the called function which is to add two numbers a and b. The numbers are the input provided by the user .After adding the result is stored in a variable c and it is printed.

**1.3 NOTATIONS**

**1.3.1 Pseudo Code**

Pseudo –False.

Code- Set of Instructions.

Pseudo code means a short, readable set of instructions written in English to explain an algorithm.

***Rules for writing Pseudo code***

* Write one statement per line
* Capitalize keywords.
* Indent to hierarchy.
* Keep statements language independent.

***Common keywords used in writing a Pseudo code***

Comment: //

Start: BEGIN

Stop: END

Input: INPUT, GET, READ

Calculate: COMPUTE, CALCULATE, ADD, SUBTRACT, INITIALIZE

Output: OUTPUT, PRINT, DISPLAY

Selection: IF, ELSE, ENDIF

Iteration: WHILE, ENDWHILE, FOR, ENDFOR

Example: Pseudo code to Add two numbers

BEGIN

GET a, b

ADD c=a+b

PRINT c

END

***Advantages:***

* Pseudo code is program independent.
* It is easily understandable by layman.
* There is no Standard syntax.
* Pseudo code cannot be compiled or executed.
* It is easy to translate pseudo code into a programming language.
* It can be easily modified as compared to flowchart.

***Disadvantages:***

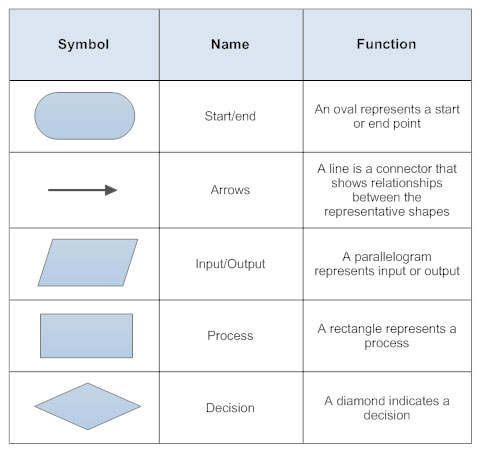
* It is not visual.
* We do not get a picture of the design.
* There is no standardized style or format.
* For a beginner, it is more difficult to follow the logic or write pseudo code as compared to flowchart.

**Flow Chart**

Flow chart is defined as the graphical or diagrammatical representation of the process of solving a problem. It represents the steps of a process in a sequential order.

***Rules:***

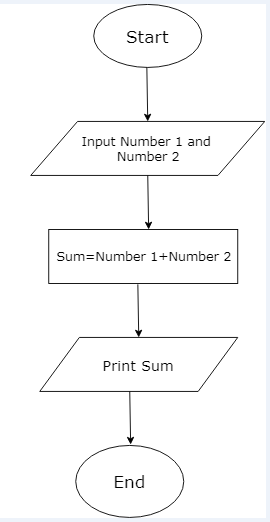
* The flowchart should be clear, neat and easy to follow.
* The flowchart must have a logical start and finish.
* Only one flow line should come out from a process symbol
* Only one flow line should enter a decision symbol. However, two or three flow lines may leave the decision symbol
* Only one flow line is used with a terminal symbol.
* Within standard symbols, write briefly and precisely.
* Intersection of flow lines should be avoided.



**Fig 1.1 Symbols used in Flow chart**

***Advantages:***

* Flowcharts help in communicating the logic of a program in a better way.
* The problems can be analyzed effectively with the help of flowchart.
* Flowcharts serve as good program documentation.
* The flowchart is ablueprint for the programmer to code efficiently.
* Flowcharts help in debugging process.



**Fig 1.2 Flowchart for adding two numbers**

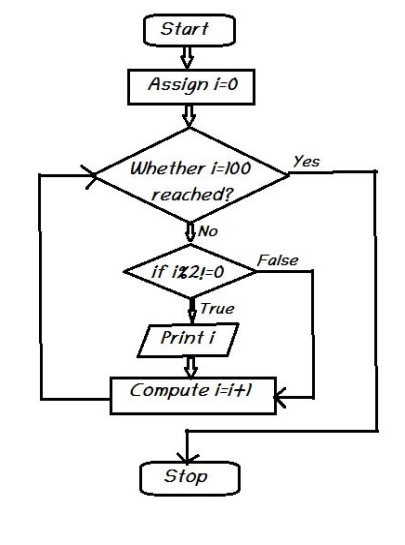
***Disadvantages:***

* The flowcharts are complex and clumsy when the program is large or complicated.
* Altering or modifying the flowcharts may require re-drawing completely.
* The cost and time taken to draw a flowchart for larger applications are expensive.

Example: Add two numbers

The above flowchart is used for adding two numbers Number1 and Number2.The numbers are the input provided by the user .After adding, the result is stored in the variable Sum and is printed.

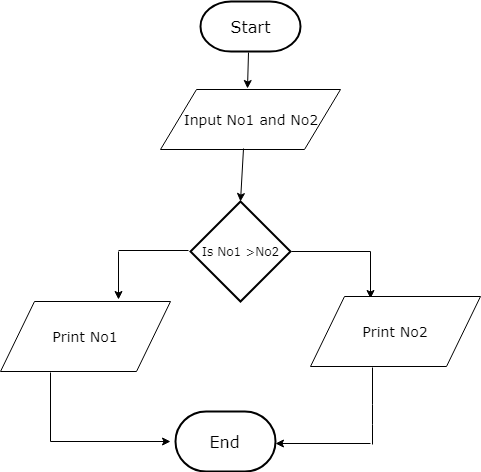
Example: Print odd numbers from 1 to 100



**Fig 1.3 Flowchart to Print odd numbers from 1 to 100**

The above flowchart is used for printing all odd numbers up to 100. The first value, i is initialized as zero. A loop is initialized. Starting from i=0, each value is divided by 2 and the remainder is captured .This operation is called modulo of a number. The modulo of first value is calculated if the result is not equal to zero then the number is printed and the number is incremented. The i value is checked with the limit, 100. The numbers are printed until the value is less than the input value.

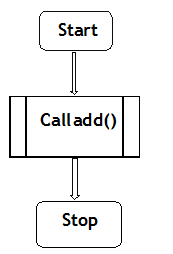
Example: Find the largest of two numbers



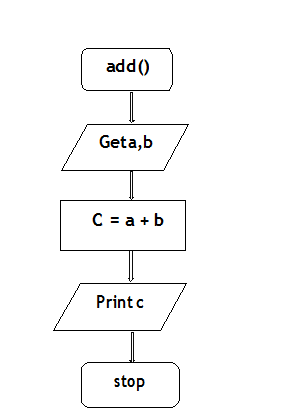
**Fig 1.4 Flowchart to find the largest of two numbers**

The above flowchart is to find the Largest of two numbers NO1 and NO2.The numbers are the input provided by the user .The number NO1 and NO2 are compared, If NO1 is larger, the number NO1 is printed or if NO2 is larger, the number NO2 is printed.

Example: Add two numbers using Function



**Fig 1.5a Flowchart of a calling function**

****

**Fig 1.5b Flowchart of a Sub function to add two numbers**

The flowchart 1.5a is for the calling function. This function is also called as Main function . The flowchart1.5b is called as the called function which is to add two numbers a and b.The numbers are the input provided by the user .After adding the result is stored in a variable c and it is printed.

Types of Flowcharts

* Process Flowchart
* Data Flowchart
* Business Process Modeling Diagram

Difference between Algorithm, Flowchart and Pseudo code

|  |  |  |
| --- | --- | --- |
| **Algorithm** | **Flowchart** | **Pseudo code** |
| An algorithm is a step by step procedure to solve a problem. | It is a graphical representation of algorithm | It is a language representation of algorithm. |
| User needs knowledge to write algorithm. | User does not need knowledge of program to draw or understand flowchart | User does not need knowledge of program language to understand or write a pseudo code. |

**Algorithmic Problem Solving For Socio Economic Conditions in Global Perspectives: People today have to strive hard to maintain**

Providing Food for Everybody

Step 1: Start

Step 2: Estimation of people below poverty line.

Step 3: Recommendation about the type and quantity of food from the food committee

Step 4: Decide the cost

Step 5: Generation of Revenue to meet the food demand

Step 6: Provide food through public distribution system at all levels.

Step 7: Get Feedback from beneficiaries as well as public

Step 8: Go to step 2

Step 9: Stop



**Fig 1.6 Flowchart for providing food for Everybody**

Process A: The number of people below poverty line in each village is estimated and the total of people in each district is calculated.

Process B: The food committee will recommend the type of food to be provided and also the quantity of food.

Process C: From the above inputs, the overall cost is calculated.

Process D: The Finance committee decides how to generate the revenue needed.

Process E: Revenue is generated by implementing Tax, Educational Cess, Import /Export duties etc.

Process F: The distribution of food is carried out through public distribution system.

Process G: Feedback is provided by the public and beneficiaries which again is an input to the process to improve the quality of the food provision

**Simple Strategies for Developing Algorithms (Iteration, Recursion)**

Iterations:

A sequence of statements is executed until a specified condition is true is called iterations.

1. For loop

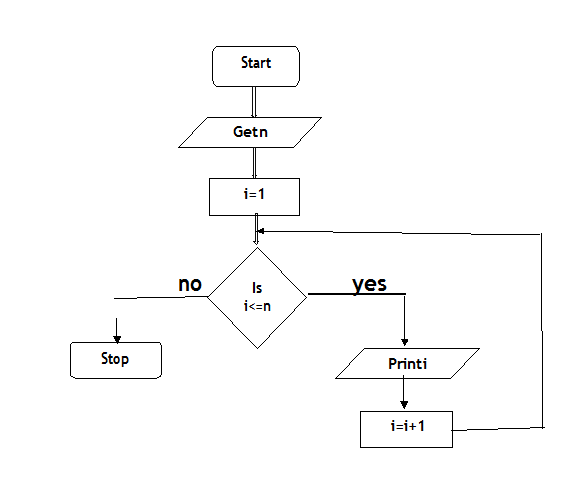
2. While loop

Example: Print n Natural numbers using FOR Loop

|  |  |
| --- | --- |
| **Algorithm** | **Pseudo code** |
| Step 1: Start  Step 2: Get the value of n.  Step 3: Initialize i=1  Step 4: if (i<=n) go to step 5 else go to step 7  Step 5: Print i value and increment i value by 1  Step 6: Go to step 4  Step 7: Stop | BEGIN  GET n  INITIALIZE i=1  FOR (i<=n) DO  PRINT i  i=i+1  ENDFOR  END |

Example: Print n Natural numbers using WHILE Loop

|  |  |
| --- | --- |
| **Algorithm** | **Pseudo code** |
| Step 1: Start  Step 2: Get the value of n.  Step 3: Initialize i=1  Step 4: if (i<=n) go to step 5 else go to step 7  Step 5: Print i value and increment i value by 1  Step 6: Go to step 4  Step 7: Stop | BEGIN  GET n  INITIALIZE i=1  WHILE(i<=n) DO  PRINT i  i=i+1  ENDWHILE  END |



**Fig 1.8 Flowchart to print n Natural numbers using WHILE Loop**

The above flowchart is to print all natural numbers up to n .The user provides the input value n. The first value, i is initialized as one. A loop is initialized. The i value is checked with n, the user input. The numbers are printed until the value is less than the user input value. When the condition becomes false the loop terminates.

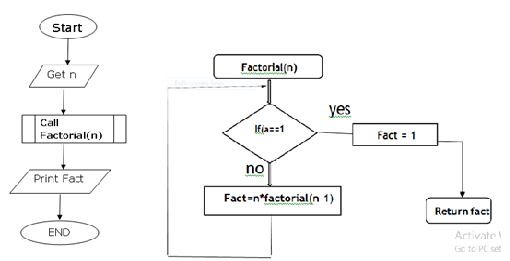
Recursion:

A function that calls itself is known as recursion.

Recursion is a process by which a function calls itself repeatedly until some specified condition has been satisfied.

Example: Factorial

|  |  |
| --- | --- |
| **Algorithm** | **Pseudo code** |
| Main Function()  Step1: Start  Step2: Get n  Step3: call factorial(n)  Step4: print fact  Step5: Stop | Main Function()  BEGIN  GET n  CALL factorial(n)  PRINT fact  END |
| Sub function factorial(n)  Step1: if(n==1) then fact=1 return fact  Step2: else fact=n\*factorial(n-1) and return fact | Sub function factorial(n)  IF(n==1) THEN  fact=1 RETURN fact  ELSE  RETURN fact=n\*factorial(n-1) |

****

**Fig 1.9 a) Flowchart of a calling function b) Flowchart of a Sub function to find factorial of a given number**

The flowchart 1.9a is to call the function factorial. This function is called as Main function or calling function. The flowchart1.5b is called as the called function which is to find the factorial of the number passed from the main function. The loop exists until the n value becomes one. Factorial is a recursive function where the function itself is called again and again.

**Difference between Iteration and Recursion**

| **Property** | **Iteration** | **Recursion** |
| --- | --- | --- |
| **Definition** | A set of instructions repeatedly executed. | Function calls itself. |
| **Application** | For loops. | For functions. |
| **Termination** | When the termination condition for the iterator ceases to be satisfied. | Through base case, where there will be no function call. |
| **Usage** | Used when time complexity needs to be balanced against an expanded code size. | Used when code size needs to be small, and time complexity is not an issue. |
| **Code Size** | Larger Code Size. | Smaller code size |
| **Time Complexity** | Relatively lower time complexity | Very high time complexity. |

**Efficiency of Algorithms: Analysis of an Algorithm**

The following are some of the factors which helps in an analyzing an algorithm

* Efficiency
* Simplicity
* Generality
* Range of Growth
* Computing order of growth

The most important factor of all these is efficiency.

Efficiency or Performance Analysis

1. Amount of time required to execute an algorithm(Time Complexity)
2. Amount of storage required by an algorithm(Space Complexity)

**Time Complexity**

* Time complexity of an algorithm is the total time required by the algorithm to run.
* To compute time we require two components – Variable part and fixed part.
* Variable Part - Run time
* Fixed Part - Compile time

Time complexity can be represented as a numerical function T(n), where T(n) can be measured as the number of steps, provided each step consumes constant time.

**Calculation of Time Complexity**

We use step count method to calculate the time complexity of an algorithm.

For example

int sum(int a[],int n)

{

Int sum=0

For (int i=0;i<n;i++)

{

Sum=sum+a[i]

}

Return sum

}

|  |  |
| --- | --- |
| **Instruction** | **Step count** |
| int sum(int a[],int n) | 0 |
| { | 0 |
| Int sum=0 | 1 |
| For (int i=0;i<n;i++) | n+1 |
| { |  |
| Sum=sum+a[i] | N |
| } | 0 |
| Return sum | 1 |
| } | 0 |
| **Total steps=1+n+1+n+1=2n+3** | |

**Space Complexity:** Space complexity of an algorithm is the amount of memory space required by the algorithm to run.

Reason for estimating space before running the program

* To know if the memory is sufficient
* The program must be of lesser size in a multiuser system ,since there will be multiple copies of the same program

To compute the space we require two components – Variable space and fixed space.

Space complexity S(P) = C + SP,

Where C is a Constant, that is the fixed space and it denotes the amount of space taken by instructions, simple variables, constants and identifiers.

And SP is the variable space is a space dependent upon instance characteristics.

Space complexity can also be represented as the sum of instruction space, Data space and Environment stack space.

**Instruction space:** The memory space required for storing the compiled instructions. It depends on the Compiler used, compiler options and the target computer

**Data space**: The amount of memory space used by Constants, variables, arrays, structures and other data structures.

**Environment stack space**: When we use a function, we will require a stack to store the return address of the main function (Calling function). This space is the environment stack space.

**Calculation of Space complexity**

|  |  |
| --- | --- |
| Type | Size |
| bool, char, unsigned char, signed char, int8 | 1 byte |
| int16, short, unsigned short | 2 bytes |
| float, int32, int, unsigned int, long, unsigned long | 4 bytes |
| double, int64, long double | 8 bytes |

Examples:

{

int c= a+b

return c

}

In the above example, variables a, b, and c are all integer types, so they will take up 4 bytes each, so total memory requirement will be (4(3) + 4) = 20 bytes, the additional 4 bytes is for return value. Since this space requirement is fixed for the above example, it is called Constant Space Complexity.

int sum(int a[],int n)

{

Int sum=0

For (int i=0;i<n;i++)

Sum=sum+a[i]

Return sum

}

* In the above example, 4\*n bytes of space is required for the array a[] n elements.
* 4 bytes each for sum, n, i and the return value.

Hence the total memory requirement will be (4n + 12), which is increasing linearly with the increase in the input value n, hence it is called as Linear Space Complexity.

**Case Studies**

* Networked Healthcare
* Clean Water for All
* Generating Energy at Low Costs
* Smart Cities
* Education for all
* Tower of Hanoi Problem

**QUESTIONS :**

1. Write an algorithm and flowchart for solving quadratic equation (of the form ax2+ bx + c).
2. M1,M2, M3 are the marks scored in 3 tests. Write an algorithm to find the average of best 2 marks.
3. Write an algorithm and pseudo code to check whether a given integer is prime number or not.
4. Write a recursive function to add two numbers a and b.
5. Describe the algorithm for finding sum and average of n numbers.
6. Draw a flow chart for computing factorial of a given number.
7. Draw the flow chart for finding sum of first n natural numbers.
8. Draw a flow chart print all prime number between to intervals.
9. Explain the algorithm to find minimum in a list and Maximum in a list.
10. Develop an algorithm for conversion from Celsius to Fahrenheit and vice versa.