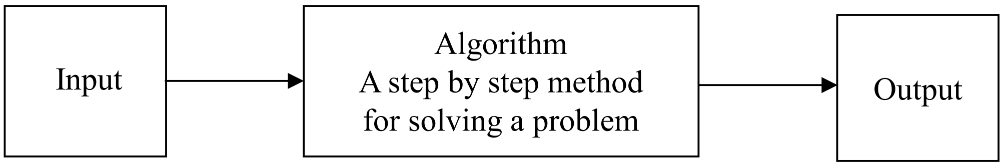
UNIT - 1

ALGORITHMIC PROBLEM SOLVING

1.1 ALGORITHM

An algorithm is defined as a step by step procedure for solving a problem. It is a ordered set of rules to solve a problem. An algorithm is a representation of a solution to a problem. It is a well-defined computational procedure consisting of a set of instructions that takes some value or set of values, as input, and produces some value or set of values, as output.



1.1.1 Properties of Algorithms

Every algorithm must have five essential properties:

(1) Inputs specified: An algorithm must have zero or more inputs, We must specify the type of the data, the amount of data, and the form that the data will take.

(2) Outputs specified: An algorithm has one or more outputs, which have a specified relation to the inputs.

(3) Definiteness: Every detail of each step must be clearly specified.

(4) Effectiveness: All operations to be performed must be sufficiently basic that they can be done exactly and in finite length.

(5) Finiteness: An algorithm must always terminate after a finite number of steps.

1.1.2 Characteristics of Algorithm Finiteness

Algorithm must terminate after a finite number of steps and further each steps must be executable in finite amount of time.

Definiteness

Instruction must be clear, user defined & Precise. There should not be any ambiguity.

Input

An algorithm has zero or more, but only finite number of inputs, zero inputs. Eg: ASCII character of 0-225.

Output

An algorithm has one or more output.

Effectiveness

An algorithm should be effective that means each of the operation to be preformed in algorithm should be computer programming language independent.

Uniqueness

Result or each steps are uniquely defined and only depend on the input and result of the preceding steps.

Generality

Algorithm should apply to set of input, rather than single input.

Feasibility

It must be possible to perform each instructions.

1.1.3 Method for Developing an Algorithm

(1) Define the problem: State the problem to be solved in clear and concise manner.

(2) List the inputs and outputs

(3) Describe the steps needed to convert input to output

(4) Test the algorithm: Choose input data and verify that the algorithm works.

1.2 BUILDING BLOCKS OF ALGORITHM

1.2.1 Statements/Instructions

The statements/instructions enable us to manipulate the data items. The instructions in Python or indeed in any high-level language are designed as components for algorithmic problem solving rather than as one-to-one translations of the underlying machine language instruction set of the computer. Thus, they allow the programmer to work at a higher level of abstraction.

Input/output statements make up one type of statement. An input statement collects a specific value from the user for a variable within the program. An output statement writes a message or the value of a program variable to the user’s screen. Assignment statement, which assigns a value to a program variable. This is similar to what an input statement does, except that the value is not collected directly from the user, but is computed by the program. Control statements, the third type of statement, affect the order in which instructions are executed. A program executes one instruction or program statement at a time. Without directions to the contrary, instructions are executed sequentially from first to last in the program. Control statements direct the flow of control.

1.2.2 State

Stored data are regarded as part of the internal state of the entity performing the algorithm. The state is stored in one or more data structures. For such computational process, the algorithm must be rigorously defined and it must be specified in the way that it applies in all possible circumstances. Data Structure specify the arrangement of data in a particular format.

1.2.3 Control Flow

Control flow (or flow of control) is the order in which individual statements, instructions or function calls of a program are executed or evaluated. At the level of machine language or assembly language, control flow instructions usually work by altering the program counter. For some central processing units (CPUs), control flow instructions available are conditional or unconditional branch instructions, also termed jumps.

kinds of control flow statements

- conditional branch
- loop - the same as conditional branch
- subroutines, co-routines, and continuations
- unconditional halt

1.2.4 Function

A function is a sequence of instructions that perform a task, bundled as a unit. Functions can accept input arguments and produce output values. A function in Python is defined by using the keyword def, after which the name of the function follows, terminated by a pair of braces (which may or may not contain input parameters) and, finally, a colon (:) signals the end of the function definition line. Immediately afterwards, indented by four spaces, we find the body of the function, which is the set of instructions that the function will execute when called.

Reason for using Function

- Hide the implementation details from their users.
- Reduce code duplication in a program.
- Help in splitting a complex task into smaller blocks, each of which becomes a function.
- Improve traceability.
- Improve readability.

1.2.5 Basic building blocks of Algorithm

Three basic building blocks of algorithm are Sequence, Selection, and Iteration.

Building Block	**Common name**	**Description**
Sequence	Action	Instructions are executed in sequential order (from first to last).
Selection	Decision or Branching	A Decision is making a choice among several actions (condition checking).
Iteration	Repetition or Loop	A Loop is one or more instructions that the computer performs repeatedly.

Sequence Structure

This is the most common form of control structure. Each statement in the program is executed one after another in the given order. The flow of sequence structure is shown in Figure.1.1.

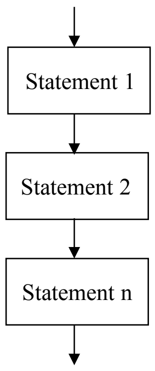


Figure 1.1. Sequence structure.

An algorithm to add two numbers entered by user

```
Step 1: Start
Step 2: Declare variables num1, num2 and sum.
Step 3: Read values num1 and num2.
Step 4: Add num1 and num2 and assign the result to sum.
          sum←num1+num2
Step 5: Display sum
Step 6: Stop
```

Selection Structure

The selection control structure is the presentation of a condition and the choice between two (or sometimes more) actions. The choice made depends on whether the condition is true or false. It is also called a *decision*, one of the three basic logic structures in computer. This structure is sometimes referred to as an if-then-else because it directs the program to perform in this way: If Condition A is True then perform Action X else perform Action Y. The flow of sequence structure is shown in Figure.1.2.

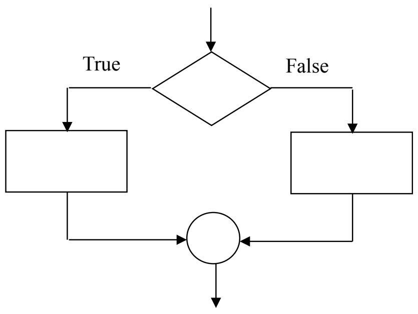


Figure 1.2. Selection structure.

Algorithm to find the largest among three different numbers entered by user.

```
Step 1: Start
Step 2: Declare variables a,b and c.
Step 3: Read variables a,b and c.
Step 4: If a>b
          If a>c
          Display a is the largest number.
          Else
          Display c is the largest number.
          Else
          If b>c
          Display b is the largest number.
          Else
          Display c is the greatest number.
Step 5: Stop
```

Algorithm to find all roots of a quadratic equation ax2+bx+c=0.

```
Step 1: Start
Step 2: Declare variables a, b, c, D, x1, x2, rp and ip;
Step 3: Calculate discriminant
          $D \leftarrow b2-4ac$
Step 4: If $D \geq 0$
          $r1 \leftarrow (-b+\sqrt{}D)/2a$
          $r2 \leftarrow (-b-\sqrt{}D)/2a$
          Display r1 and r2 as roots.
          Else
          Calculate real part and imaginary part
          $rp \leftarrow b/2a$
          $ip \leftarrow \sqrt{}(-D)/2a$
          Display rp+j(ip) and rp-j(ip) as roots
Step 5: Stop
```

Iteration Structure

The iterative process is also called as repetition or looping. It carries out a particular action over and over again until some condition is met. A loop is created to return the program to where the repetition has started for as long as it takes until the condition is met.

There are two ways of testing to see if the condition is met: pre-test loops, and post-test loops. In **pre-test** loops, the condition is tested at the beginning of the loop. If the condition fails, the looping process will never carried out. Here the loop is operated only when condition is met. Pre-test loop ends when the condition fails.

Post-test loops test the condition at the end of the loop. The looping process is executed first whether the condition is true or false and will keep repeating until the condition is met. This loop ends when the condition is true. The structures of pre-tested and post-tested loops are shown in Figure.1.3.

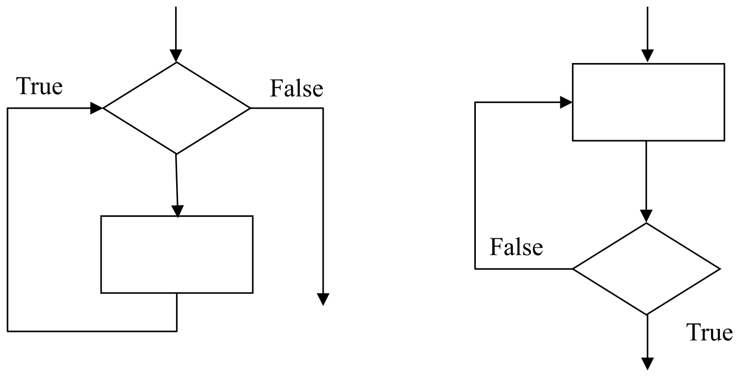


Figure 1.3. Iteration structure.

Algorithm to find the factorial of a number entered by user

```
Step 1: Start
Step 2: Declare variables n,factorial and i.
Step 3: Initialize variables
        factorial←1
        i←1
Step 4: Read value of n
Step 5: Repeat the steps until i=n
        5.1: factorial←factorial*i
        5.2: i←i+1
Step 6: Display factorial
Step 7: Stop
```

An algorithm to find the Fibonacci series till term≤1000

```
Step 1: Start
Step 2: Declare variables first_term,second_term and temp.
Step 3: Initialize variables first_term←0 second_term←1
Step 4: Display first_term and second_term
Step 5: Repeat the steps until second_term≤1000
        5.1: temp←second_term
        5.2: second_term←second_term+first term
        5.3: first_term←temp
        5.4: Display second_term
Step 6: Stop
```

1.3 ALGORITHM NOTATIONS (EXPRESSING ALGORITHMS)

An algorithm is a sequence of finite instructions, often used for calculation and data processing. Algorithms can be expressed in many kinds of notation, including

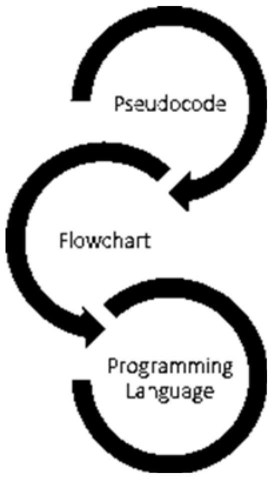


Fig 1.3.1 Notations

1.3.1 PSEUDOCODE

Pseudocode (pronounced SOO-doh-kohd) is a kind of structured English for representing algorithms and is a "text-based" detail design tool for human understanding rather than machine understanding. It allows the designer to concentrate on the logic of the algorithm without worrying about the details of language syntax.

Once the pseudo code is prepared, it is rewritten using the syntax and semantic rules of a programming language.

Rules of Pseudo code

- Start with an algorithm and phrase it using words that are easily transcribed into computer instructions.
- Indent when you are enclosing instructions within a loop or a conditional clause.
- Avoid words associated with a certain kind of computer language.
- Do not include data declarations in pseudo code.

Three basic constructs for flow of control

1. Sequence
   - Default mode.
   - Used for the sequential execution of statements, one line after another.
2. Selection
   - Used for decisions to choose between two or more alternative paths.
3. Repetition
   - Used for looping to repeat a piece of code several times.

Sequence

It is indicated by writing one statement after another, each statement on a line by itself, and all statements aligned with the same indent. The actions are performed in the order in which they are written, from top to bottom.

Common Keywords

Input: READ, INPUT, OBTAIN, GET

Output: PRINT, OUTPUT, DISPLAY, SHOW

Compute: COMPUTE, CALCULATE, DETERMINE

Initialize: SET, INIT

Add one: INCREMENT, BUMP

Examples

(i) Pseudo code for computing the area of a rectangle.

```
READ height of rectangle
READ width of rectangle
COMPUTE area as height times width
PRINT area
```

(ii) Pseudo code for computing the average of five numbers.

```
PRINT "Enter 5 numbers"
READ n1, n2, n3, n4, n5
PRINT "The average is"
SET avg to (n1+n2+n3+n4+n5)/5
PRINT avg
```

Selection

It is a decision (selection) in which a choice is made between two alternative courses of action and it comprises of the following constructs:

- **IF-THEN-ELSE**
- **CASE...ENDCASE**

IF-THEN-ELSE

Binary choice on a given Boolean condition is indicated by the use of four keywords:

◦ IF, THEN, ELSE, and ENDIF.

The general form is:

```
IF condition THEN
    sequence 1
ELSE
    sequence 2
ENDIF
```

The ELSE keyword and “sequence 2” are optional. If the condition is true, sequence 1 is performed, otherwise sequence 2 is performed.

Examples

(i) Pseudo code to check whether the number is odd or even.

```
READ number
IF number MOD 2 = 0 THEN
    DISPLAY “Number is Even”
ELSE
    DISPLAY “Number is Odd”
ENDIF
```

(ii) Pseudo code to check whether the given non-zero number is positive or negative.

```
READ number
IF num is less than 0 THEN
PRINT num is negative
ELSE
PRINT num is positive
ENDIF
```

CASE

CASE is a multiway branch (decision) based on the value of an expression. CASE is a generalization of IF-THEN-ELSE. Four keywords, CASE, OF, OTHERS, and ENDCASE, and conditions are used to indicate the various alternatives.

The general form is:

```
CASE expression OF
    condition 1 : sequence 1
    condition 2 : sequence 2
    ...
    condition n : sequence n
    OTHERS:
    default sequence
ENDCASE
```

The OTHERS clause with its default sequence is optional. Conditions are normally numbers or characters indicating the value of "expression", but they can be English statements or some other notation that specifies the condition under which the given sequence is to be performed. A certain sequence may be associated with more than one condition.

Examples

(i) Pseudo code for simple calculator

```
READ n1, n2
READ choice
CASE choice OF
    +  : PRINT n1+n2
    –  : PRINT n1-n2
    *  : PRINT n1*n2
    /  : PRINT n1/n2
ENDCASE
```

(ii) Pseudo code for determining gradepoints from grades.

```
READ grade
CASE  grade  OF
    S  : gradepoint = 10
    A  : gradepoint = 9
```

```
        B   : gradepoint = 8
        C   : gradepoint = 7
        D   : gradepoint = 6
        E   : gradepoint = 5
        U   : gradepoint = 0
    ENDCASE
    DISPLAY gradepoint
```

Repetition

It is a loop (iteration) based on the satisfaction of some condition(s). It comprises of the following constructs:

- WHILE...ENDWHILE
- REPEAT...UNTIL
- FOR...ENDFOR

WHILE...ENDWHILE

It is a loop (repetition) with a simple condition at its beginning. The beginning and ending of the loop are indicated by two keywords WHILE and ENDWHILE.

The general form is:

```
WHILE condition
    sequence
ENDWHILE
```

The loop is entered only if the condition is true. The "sequence" is performed for each iteration. At the conclusion of each iteration, the condition is evaluated and the loop continues as long as the condition is true.

Examples

(i) Pseudo code to print the numbers from 1 to 100.

```
n=1
WHILE n is less than or equal to 100
    DISPLAY n
    INCREMENT n by 1
ENDWHILE
```

(ii) Pseudo code to print the sum of the digits of a given number

```
INPUT a Number
INITIALIZE Sum to zero
WHILE Number is not zero
    COMPUTE Remainder by Number Mod 10
    ADD Remainder to Sum
    DIVIDE Number by 10
ENDWHILE
PRINT Sum
```

REPEAT...UNTIL

It is a loop with a simple condition at the bottom. This loop is similar to the WHILE loop except that the test is performed at the bottom of the loop instead of at the top. Two keywords, REPEAT and UNTIL are used.

The general form is:

```
REPEAT
    sequence
UNTIL condition
```

The "sequence" in this type of loop is always performed at least once, because the test is peformed after the sequence is executed. At the conclusion of each iteration, the condition is evaluated, and the loop repeats if the condition is false. The loop terminates when the condition becomes true.

Examples

(i) Pseudo code to print the numbers from 1 to 100.

```
n=1
REPEAT
    DISPLAY n
    INCREMENT n by 1
UNTIL n is greater than 100
```

(ii) Pseudo code to print the sum of the digits of a given number

```
INPUT a Number
INITIALIZE Sum to zero
```

```
REPEAT
    COMPUTE Remainder by Number Mod 10
    ADD Remainder to Sum
    DIVIDE Number by 10
UNTIL Number is zero
PRINT Sum
```

FOR...ENDFOR

FOR is a "counting" loop. This loop is a specialized construct for iterating a specific number of times, often called a "counting" loop. Two keywords, FOR and ENDFOR are used.

The general form is:

```
FOR iteration bounds
    sequence
ENDFOR
```

Examples

(i) Pseudo code to print the numbers from 1 to 100.

```
FOR n=1 to 100
    DISPLAY n
ENDFOR
```

(ii) Pseudo code to input ten numbers and print the sum.

```
INITIALIZE sum to 0
FOR n=1 to 10
    INPUT number
    COMPUTE sum as sum+number
ENDFOR
DISPLAY sum
```

Nested Constructs

The constructs can be embedded within each other, and this is made clear by use of indenting. Nested constructs should be clearly indented from their surrounding constructs.

Examples

(i) Pseudo code to find the smallest among three numbers using nested IF construct.

```
READ the three numbers a, b, c.
```

```
IF a is less than b THEN
    IF a is less than c THEN
            PRINT a,
    ELSE
            PRINT c
    ENDIF
ELSE IF b is less than a THEN
    IF b is less than c THEN
            PRINT b
    ELSE
            PRINT c
    ENDIF
ENDIF
```

(ii) Pseudo construct using IF construct nested within REPEAT construct.

```
SET total to zero
REPEAT
READ Temperature
IF Temperature > Freezing THEN
    INCREMENT total
ENDIF
UNTIL Temperature < zero
PRINT total
```

Invoking Subprocedures

To call subprocedures (functions), CALL keyword is used with the following structure:

CALL subprocedurename [**WITH** argumentslist] [**RETURNING** returnvalue]

Here, "**WITH** argumentslist" and "**RETURNING** returnvalue" are optional. "**WITH** argumentslist" is used to call a function with arguments. "**RETURNING** returnvalue" is used when a called function returns a value.

Examples:

CALL SumAndAvg **WITH** NumberList

- Calls a function SumAndAvg with NumberList as an argument. This function does not return any value.

CALL SwapItems **WITH** CurrentItem and TargetItem

- Calls a function SwapItems with CurrentItem and TargetItem as arguments. This function does not return any value.

CALL getBalance **RETURNING** BalanceAmt

- Calls a function getBalance which returns BalanceAmt as its return value. This function does not take any argument.

Advantages of Pseudo code

(1) Can be read and understood easily.

(2) Can be done easily on a word processor.

(3) Can be modified easily.

(4) Implements structured concepts well.

(5) Clarifies algorithms in many cases.

(6) Imposes increased discipline on the process of documenting detailed design.

(7) Provides additional level at which inspection can be performed.

(8) Helps to trap defects before they become code.

(9) Increases product reliability.

(10) Converting a pseudocode to a program is simple.

Disadvantages of Pseudo code

(1) Creates an additional level of documentation to maintain.

(2) Introduces error possibilities in translating to code.

(3) May require tool to extract pseudocode and facilitate drawing flowcharts.

(4) There is no standardized format, so one pseudocode may be different from another.

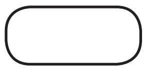
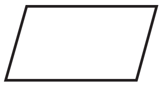
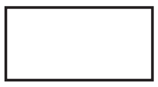
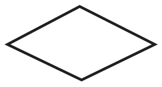
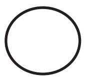
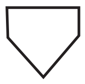
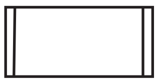
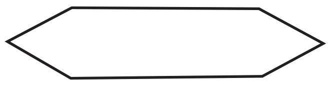
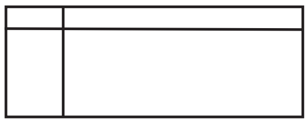
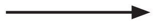
(5) For a beginner, it is more difficult to follow the logic and write pseudocode as compared to flowchart.

(6) We do not get a picture of the design.

1.3.2 Flowchart

Flowchart is a diagrammatic representation of an algorithm or a stepwise process, showing the steps as boxes of various kinds, and their order by connecting these with arrows. Flowcharts are used in designing or documenting a process or program. In other words, a flow chart, or flow diagram, is a graphical representation of a process or system that details the sequencing of steps required to create output. A flowchart is a picture of the separate steps of a process in sequential order.

Flowchart is very helpful in writing program and explaining program to others. Flowchart makes us to understand the problem unambiguously. It is often used by programmer as a program planning tool for organizing a sequence of step necessary to solve a problem by a computer.

Symbol	Purpose	Description
	Terminal(Stop/Start)	Represents start and end of flowchart.
	Input/output	Represents input and output operation.
	Processing	Represents arithmetic operations and data-manipulations.
	Decision	Represents the decision making operation in which there are two alternatives, true and false.
	On-page Connector	Used to join different flow line
	Off-page Connector	Used to connect flowchart portion on different page.
	Predefined Process/ Function	Represent a group of statements performing one processing task and for function call.
	Looping statements	Used for iterative statements(Looping structure).
	Declare	Used for declaration statements
	Flow line	Indicates the flow of logic by connecting symbols.

GUIDELINES

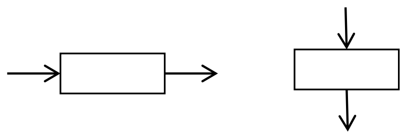
The following are some guidelines in flowcharting:

(a) In drawing a proper flowchart, all necessary requirements should be listed out in logical order.

(b) The flowchart should be clear, neat and easy to follow. There should not be any room for ambiguity in understanding the flowchart.

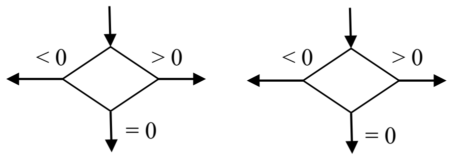
(c) The usual direction of the flow of a procedure or system is from left to right or top to bottom.

(d) Only one flow line should come out from a process symbol.

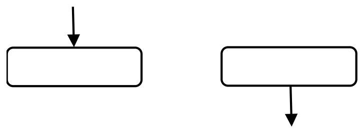


or

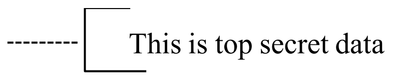
(e) Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, should leave the decision symbol.



(f) Only one flow line is used in conjunction with terminal symbol.



(g) Write within standard symbols briefly. As necessary, you can use the annotation symbol to describe data or computational steps more clearly.



(h) If the flowchart becomes complex, it is better to use connector symbols to reduce the number of flow lines. Avoid the intersection of flow lines if you want to make it more effective and better way of communication.

(i) Ensure that the flowchart has a logical *start* and *finish.*

(j) It is useful to test the validity of the flowchart by passing through it with a simple test data.

Advantages of flow charts

- It is easy to understand
- A problem can be analysed easily with flowchart
- It gives clear idea of a program
- It acts as a guide during the program development
- It helps to clear the errors in coding
- It helps in maintenance of code

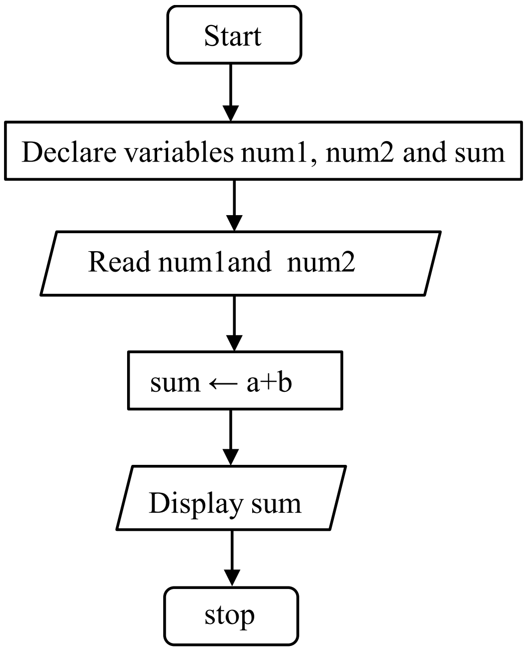
Disadvantages of flow charts

- It cannot be prepared for difficult programs
- Alterations and modifications cannot be done easily
- It is not typed, so its preparation is little difficult

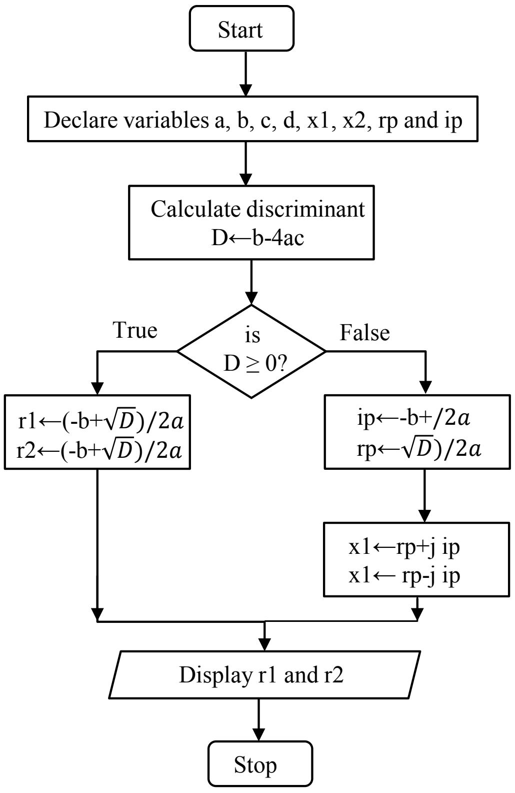
Rules for drawing a flow chart is given below

- The standard symbols should only be used.
- The arrowheads in the flowchart represent the direction of flow of control in the problem.
- The usual direction of the flow of procedure is from top to bottom or left to right.
- The flow lines should not cross each other.
- Be consistent in using names and variables in the flowchart.
- Keep the flowchart as simple as possible.
- Words in the flowchart symbols should be common statements and easy to understand.
- Chart main line of logic, and then incorporate all the details of logic.
- If a new page is needed for flowcharting, then use connectors for better representation.
- Don’t chart every details or the flowchart will only be graphical represented

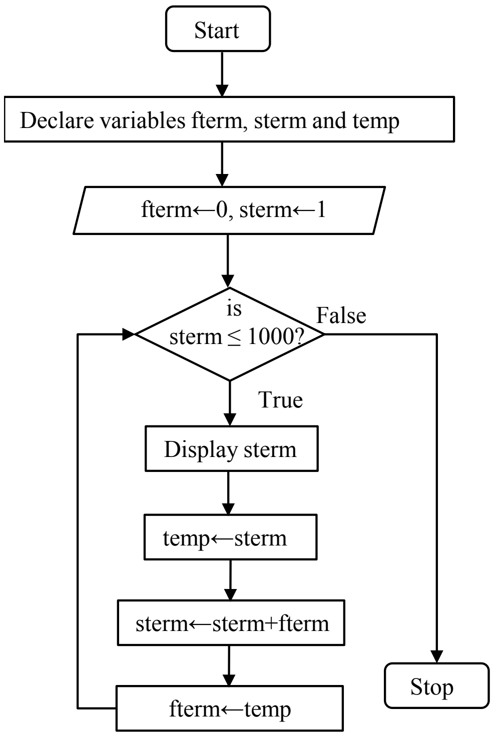
Flowchart representation to add two numbers



Flowchart representation to find all the roots of a quadratic equation ax^2+bx+c=0



Flowchart representation to find the Fibonacci series till term≤1000



1.3.3 Programming Language

Representation of Algorithm using Programming Language

Algorithms describe the solution to a problem in terms of the data needed to represent the problem instance and the set of steps necessary to produce the intended result. Programming languages must provide a notational way to represent both the process and the data. It also provide control constructs and data types.

Programming is the process of taking an algorithm and encoding it into a notation, a programming language, so that it can be executed by a computer. Although many programming languages and many different types of computers exist, the important first step is the need to have the solution. Without an algorithm there can be no program. Control constructs allow algorithmic steps to be represented in a convenient yet unambiguous way. At a minimum, algorithms require constructs that perform sequential processing, selection for decision-making, and iteration for repetitive control. As long as the language provides these basic statements, it can be used for algorithm representation.

Programming is implementing the already solved problem (algorithm) in a specific computer language where syntax and other relevant parameters are different, based on different programming languages.

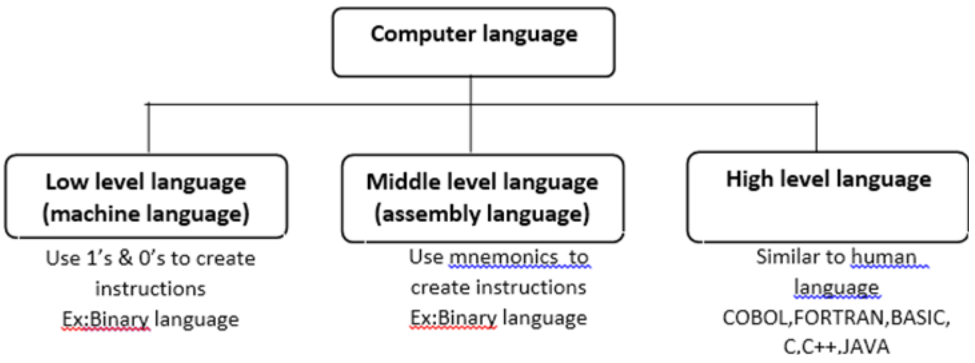

Computer language
Low level language (machine language)
Use 1's & 0's to create instructions
Ex:Binary language
Middle level language (assembly language)
Use mnemonics to create instructions
Ex:Binary language
High level language
Similar to human language
COBOL,FORTRAN,BASIC, C,C++,JAVA

Fig Programming

Low level Language(Machine level Language)

A low-level language is a programming language that deals with a computer's hardware components and constraints. In simple we can say that ,low level language can only be understand by computer processor and components. Binary and assembly languages are examples for low level language.

Middle level Language (Intermediate Language)

Medium-level language serves as the bridge between the raw hardware and programming layer of a computer system.Medium-level language is also known as intermediate programming language and pseudo language. C intermediate language and Java byte code are some examples of medium-level language.

High level Language (Human understandable Language)

A high-level language is any programming language that enables development of a program in a much more user-friendly programming context.High-level languages are designed to be used by the human operator or the programmer.They are referred to as "closer to humans." In other words, their programming style and context is easier to learn and implement than low- level languages. BASIC, C/C++ and Java are popular examples of high-level languages.

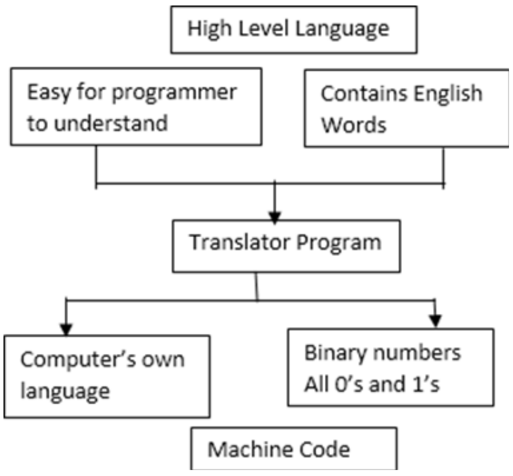


Fig 1.3.8 Programming

1.4 ALGORITHMIC PROBLEM SOLVING

1.4.1 Definition

"Algorithmic-problem solving"; this means solving problems that require the formulation of an algorithm for their solution. The formulation of algorithms has always been an important element of problem-solving .

1.4.2 An algorithmic Development Process

Every problem solution starts with a plan. That plan is called an algorithm.An algorithm is a plan for solving a problem.

Different ways in which algorithm solves Problem

- A computer is a tool that can be used to implement a plan for solving a problem.
- A computer program is a set of instructions for a computer. These instructions describe the steps that the computer must follow to implement a plan.
- An algorithm is a plan for solving a problem. A person must design an algorithm.
- A person must translate an algorithm into a computer program.

1.4.3 Fundamentals of Algorithmic Problem Solving

- **Understanding the problem :** An input to an algorithm specifies an instance of the problem the algorithm solves. It's also important to specify exactly the range of instances the algorithm needs to handle. Before this we have to clearly understand the problem and clarify the doubts after leading the problems description. Correct algorithm should work for all possible inputs.

- **Ascertaining the capabilities of a computational Device:** The second step is to ascertain the capabilities of a machine. The essence of von-Neumann machines architecture is captured by RAM, Here the instructions are executed one after another, one operation at a time, Algorithms designed to be executed on such machines are called sequential algorithms. An algorithm which has the capability of executing the operations concurrently is called parallel algorithms. RAM model doesn't support this.
- **Choosing between exact and approximate problem solving:** The next decision is to choose between solving the problem exactly or solving it approximately. Based on this, the algorithms are classified as exact and approximation algorithms. There are three issues to choose an approximation algorithm. First, there are certain problems like extracting square roots, solving non-linear equations which cannot be solved exactly. Secondly, if the problem is complicated it slows the operations. E.g. traveling salesman problem. Third, this algorithm can be a part of a more sophisticated algorithm that solves a problem exactly.
- **Deciding on data structures:** Data structures play a vital role in designing and analysing the algorithms. Some of the algorithm design techniques also depend on the structuring data specifying a problem's instance. Algorithm + Data structure = Programs
- **Algorithm Design Techniques:** An algorithm design technique is a general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing. Learning these techniques are important for two reasons, First, they provide guidance for designing for new problems. Second, algorithms are the cornerstones of computer science. Algorithm design techniques make it possible to classify algorithms according to an underlying design idea; therefore, they can serve as a natural way to both categorize and study algorithms.
- **Methods of specifying an Algorithm:** A Pseudocode , which is a mixture of a natural language and programming language like constructs. Its usage is similar to algorithm descriptions for writing psuedocode there are some dialects which omits declarations of variables, use indentation to show the scope of the statements such as if, for and while. Use → for assignment operations, (//) two slashes for comments. To specify algorithm flowchart is used which is a method of expressing an algorithm by a collection of connected geometric shapes consisting descriptions of the algorithm's steps.
- **Proving an Algorithm's correctness:** Correctness has to be proved for every algorithm. To prove that the algorithm gives the required result for every legitimate input in a finite amount of time. For some algorithms, a proof of correctness is quite easy; for others it can be quite complex. A technique used for proving correctness s by mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs. But we need one instance of its input for which the algorithm fails. If it is incorrect, redesign the algorithm, with the same decisions of data structures

design technique etc. The notion of correctness for approximation algorithms is less straightforward than it is for exact algorithm. For example, in gcd (m,n) two observations are made. One is the second number gets smaller on every iteration and the algorithm stops when the second number becomes 0.

- **Analysing an algorithm:** There are two kinds of algorithm efficiency: time and space efficiency. Time efficiency indicates how fast the algorithm runs; space efficiency indicates how much extra memory the algorithm needs. Another desirable characteristic is simplicity. Simper algorithms are easier to understand and program, the resulting programs will be easier to debug. For e.g. Euclid's algorithm to fid gcd (m,n) is simple than the algorithm which uses the prime factorization. Another desirable characteristic is generality. Two issues here are generality of the problem the algorithm solves and the range of inputs it accepts. The designing of algorithm in general terms is sometimes easier. For eg, the general problem of computing the gcd of two integers and to solve the problem. But at times designing a general algorithm is unnecessary or difficult or even impossible. For eg, it is unnecessary to sort a list of n numbers to find its median, which is its [n/2]th smallest element. As to the range of inputs, we should aim at a range of inputs that is natural for the problem at hand.
- **Coding an algorithm:** Programming the algorithm by using some programming language. Formal verification is done for small programs. Validity is done thru testing and debugging. Inputs should fall within a range and hence require no verification. Some compilers allow code optimization which can speed up a program by a constant factor whereas a better algorithm can make a difference in their running time. The analysis has to be done in various sets of inputs. A good algorithm is a result of repeated effort & work. The program's stopping / terminating condition has to be set. The optimality is an interesting issue which relies on the complexity of the problem to be solved. Another important issue is the question of whether or not every problem can be solved by an algorithm. And the last, is to avoid the ambiguity which arises for a complicated algorithm.

1.5 SIMPLE STRATEGIES FOR DEVELOPING ALGORITHMS

Step 1: Obtain a description of the problem.

Step 2: Analyze the problem.

Step 3: Develop a high-level algorithm.

Step 4: Refine the algorithm by adding more detail.

Step 5: Review the algorithm.

Step 1: Obtain a description of the problem.

This step is much more difficult than it appears. In the following discussion, the word client refers to someone who wants to find a solution to a problem, The word developer refers to someone who finds a way to solve the problem. The developer must create an algorithm that will solve the client's problem.

Step 2: Analyze the problem.

The purpose of this step is to determine both the starting and ending points for solving the problem. This process is analogous to a mathematician determining what is given and what must be proven. A good problem description makes it easier to perform this step.

Step 3: Develop a high-level algorithm.

An algorithm is a plan for solving a problem, but plans come in several levels of detail. It's usually better to start with a high-level algorithm that includes the major part of a solution, but leaves the details until later. We can use an everyday example to demonstrate a high-level algorithm.

Problem: I need to send a birthday card to my brother, Mark.

Analysis: I don't have a card. I prefer to buy a card rather than make one myself.

Example: Go to a store that sells greeting cards Select a card Purchase a card & mail the card.

Step 4: Refine the algorithm by adding more detail.

A high-level algorithm shows the major steps that need to be followed to solve a problem. The technique of gradually working from a high-level to a detailed algorithm is often called stepwise refinement. Stepwise refinement is a process for developing a detailed algorithm by gradually adding detail to a high-level algorithm.

Step 5: Review the algorithm.

The final step is to review the algorithm. First, we need to work through the algorithm step by step to determine whether or not it will solve the original problem.

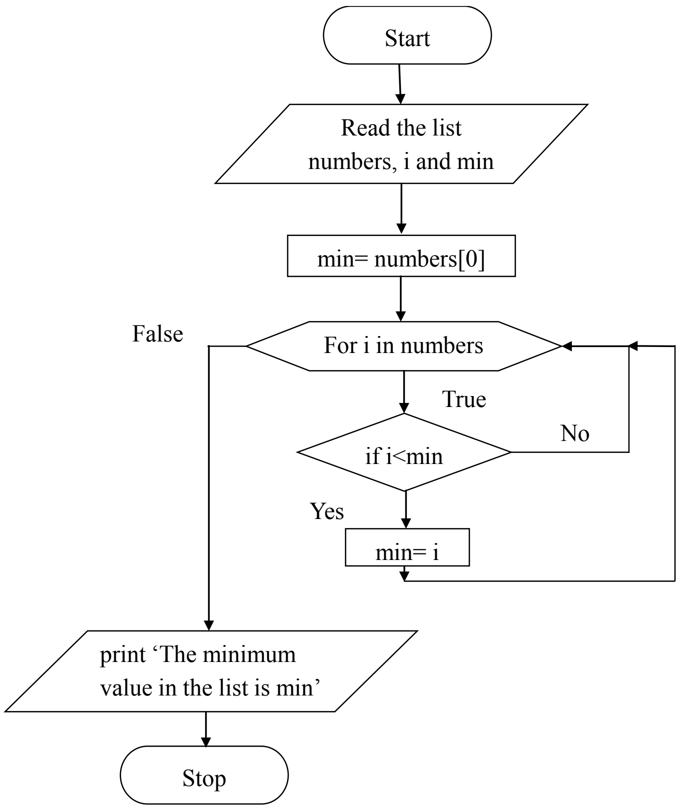
1.6 ILLUSTRATIVE PROBLEMS

1.6.1 Find minimum in a list

Algorithm:

1. Start
2. Get positive numbers from user and add it in to the List L
3. Set min to L[0].
4. For each number x in the list L, compare it to min. If x is smaller, set min to x.
5. min is now set to the minimum number in the list.
6. Stop

Flow chart for finding minimum number in a list.


Start
Read the list numbers, i and min
min= numbers[0]
False
For i in numbers
True
No
if i<min
Yes
min= i
print 'The minimum value in the list is min'
Stop

Pseudo code to find minimum in a list

```
READ the list of numbers
INITIALIZE min with the first element of the list
FOR each element in the list of numbers
    IF element is less than min
        COPY element to min
    ENDIF
ENDFOR
PRINT min as the minimum value
```

1.6.2 Pseudo code to insert a card in a list of sorted cards

Playing cards is one of the techniques of sorting and the steps are shown as follows:

- Start with an empty left hand and cards face down on the table.
- Then remove one card at a time from the table and Insert it into the correct position in the left hand.
- To find a correct position for a card, we compare it with each of the cards already in the hand from left to right.

Once the position is found, the cards from that position are moved to the next higher indexed position and in that order.

New card is inserted at the current position.

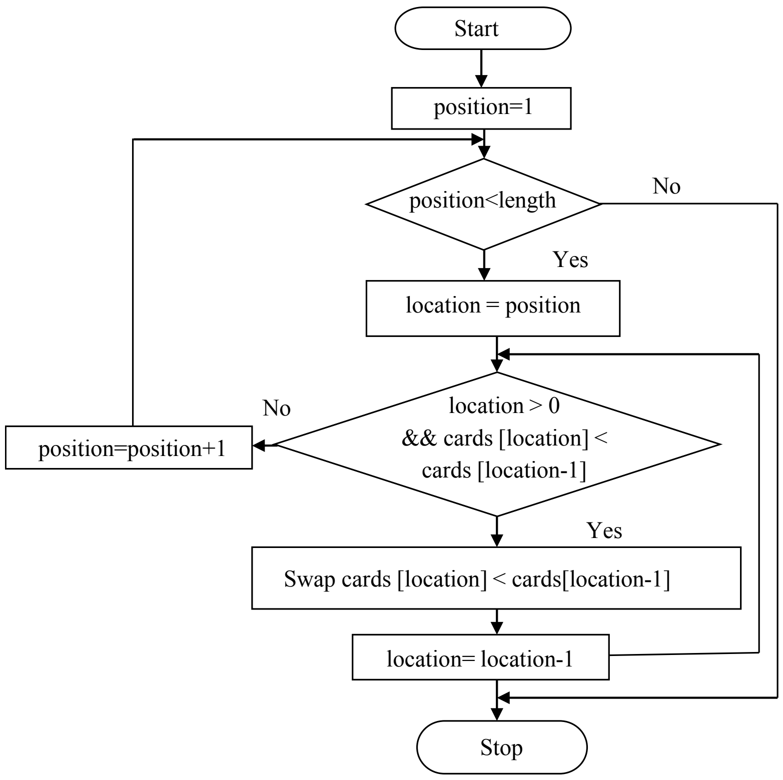
Algorithm

1. Start
2. Ask for value to insert
3. Find the correct position to insert, If position cannot be found ,then insert at the end.
4. Move each element from the backup to one position, until you get position to insert.
5. Insert a value into the required position
6. Increase array counter
7. Stop

Pseudo Code

```
READ the list of sorted cards
READ newcard
SET pos to 0
WHILE (pos < numberOfCards) and (CARDS [pos] <= newcard)
    INCREMENT pos
ENDWHILE
IF  pos < numberOfCards then
    INCREMENT numberOfCards by 1
    FOR counter = last position used to pos by -1
      SET CARDS [counter + 1] to array CARDS [counter]
    ENDFOR
ENDIF
SET CARDS [pos] to newcard
```

Flowchart for inserting a card in a list of sorted cards



1.6.3 Guessing Game

Problem: Guessing Game – guessing a number within a range of numbers.

Algorithm:

Step 1. Generate a random number in between 1 and 100.

Step 2. Prompt the user for a guess number.

Step 3. Determine if the guessing number is

Step 3. (a). Equal to the random number: If it is, output a message that says the guessing is correct and go to step 4.

Step 3. (b). Less than the random number: If it is,

(i) Output a message “Too low,”

(ii) Ask if the user wants to continue the game. If yes, ask for another guess number; otherwise go to step 4.

Step 3. (c). Greater than the random number: If it is,

(i) Output a message “Too high,”

(ii) Ask if the user wants to continue the game. If yes, ask for another guess number; otherwise go to step 4.

Step 4. Stop.

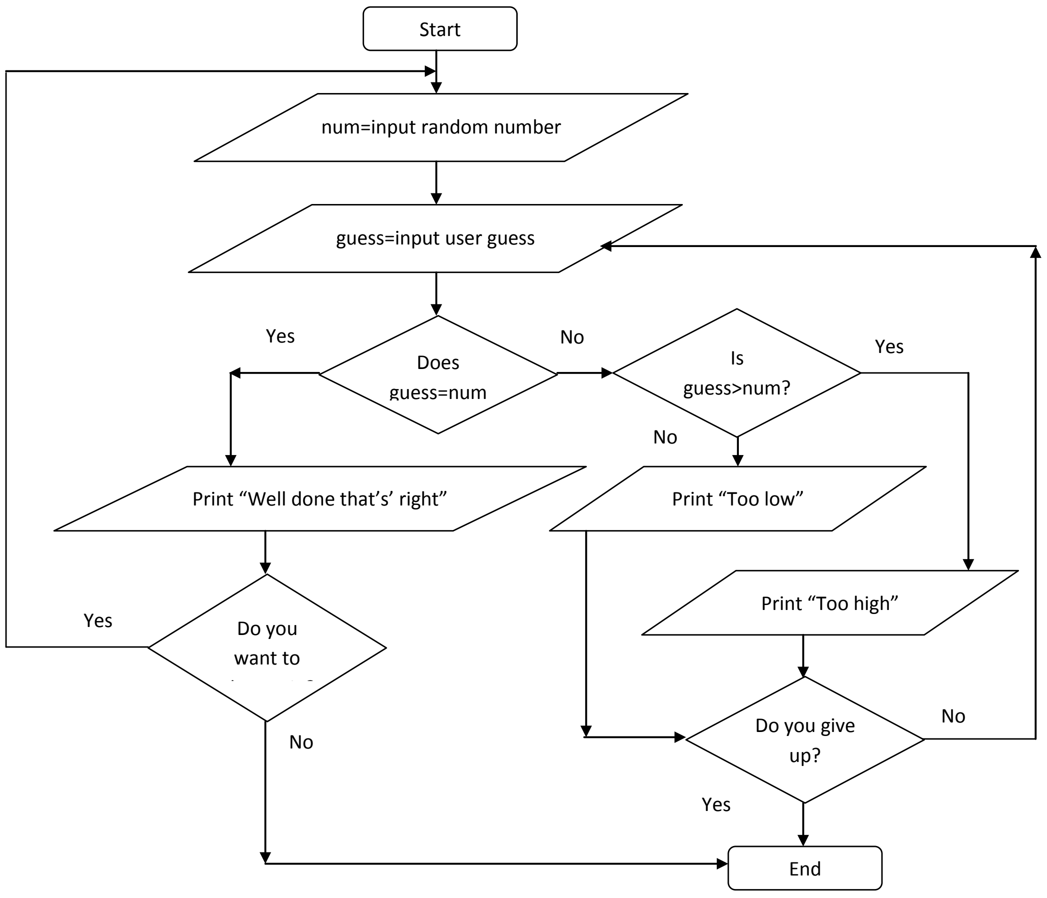
Pseudo code to guess an integer number in a range

This task allows a player to guess a number that lies between 1 and 100. The code should keep repeating until the player guesses the correct number. The number guessed by the system in the following pseudo code is 45.

Pseudo Code

```
READ number from player
WHILE number is not equal to 45
    IF number is less than 45 THEN
            PRINT Your guess is too small.
    ELSE IN number is greater than 45
            PRINT Your guess is too high.
    ENDIF
    IF number does not fall between 1 and 100 THEN
            PRINT Guess a number between 1 and 100
    ENDIF
    READ number
ENDWHILE
PRINT Your guess is correct
```

Flowchart:



1.6.4 Towers of Hanoi

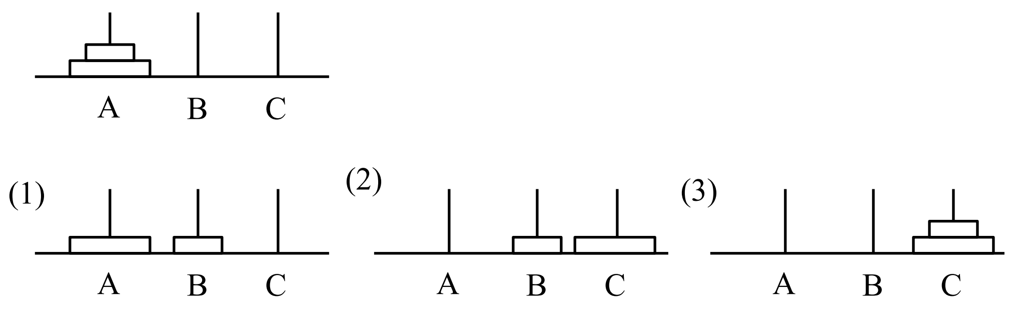
Problem:

Tower of Hanoi is a mathematical puzzle with three rods and 'n' numbers of discs. These rings are of different sizes and stacked upon in an ascending order, i.e. the smaller one sits over the larger one. The objective is to move all the disks to some another tower without violating the sequence of arrangement.

Rules to be followed:

- Only one disk can be moved among the towers at any given time.
- Only the "top" disk can be removed.
- No large disk can sit over a small disk.

2 DISKS



Recursive Algorithm of Towers of Hanoi:

Main program

```
#fr-source rod; ar-auxillary rod; tr-target rod
    Step 1: Start
    Step 2: Define function
    Step 3: Declare variable n
    Step 4: Enter the number of disc n
    Step 5: Initialize the fr=a, ar=c, tr = b
    Step 4: Call the function towerfun (n, a , c, b)
    Step 5: Stop
```

Algorithm for Function Definition

```
Step 1: if n=1
Step 1.a: Print Move disc 1 from fr to tr
Step 2: Call the function with n-1, fr, ar, tr
```

Step 3: Move disc n from fr to tr

Step 4: Call function with n-1, ar, tr, fr

Step 5: Return

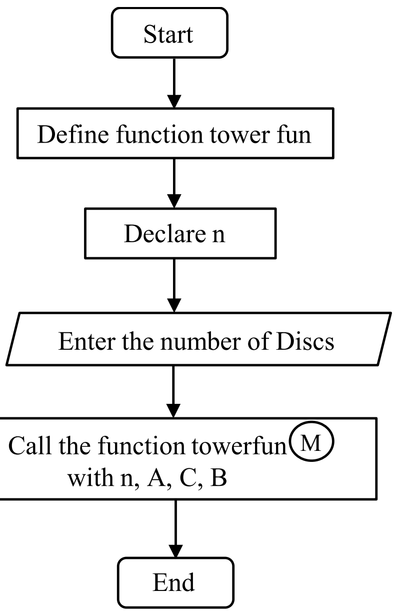
Pseudo code for Towers of Hanoi

```
SUBPROCEDURE Hanoi(disk, source, destination, auxiliary)
    IF disk == 0, THEN
        move disk from source to destination
    ELSE
        MoveTower(disk - 1, source, auxiliary, destination)
        Move disk from source to destination
        MoveTower(disk - 1, auxiliary, destination, source)
    ENDIF
END PROCEDURE
```

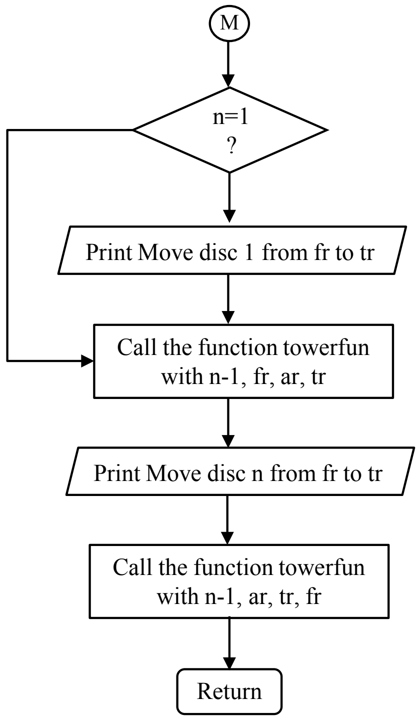
The above subprocedure "Hanoi" can be called by the following statement:

CALL Hanoi **WITH** 5, A, B, C

Flowchart: Main program


Start
Define function tower fun
Declare n
Enter the number of Discs
Call the function towerfun with n, A, C, B
M
End

Flowchart: Function towerfun()


M
n=1 ?
Print Move disc 1 from fr to tr
Call the function towerfun with n-1, fr, ar, tr
Print Move disc n from fr to tr
Call the function towerfun with n-1, ar, tr, fr
Return

Additional algorithms and programs

Algorithm to check whether a number entered by user is prime or not.

```
Step 1: Start
Step 2: Declare variables n,i,flag
Step 3: Initialize variables
        flag←1
        i←2
Step 4: Read n from user
Step 5: Repeat the steps until i<(n/2)
        5.1 If remainder of n÷i equals 0
        flag←0
        Go to step 6
        5.2 i←i+1
Step 6: If flag=0
        Display n is not prime
        else
        Display n is prime
Step 7: Stop
```

Algorithm to find minimum and maximum elements in the list.

Problem: Given a list of positive numbers, return the largest number on the list.

Inputs: A list L of positive numbers.

Outputs: A number n, which will be the largest number of the list.

Algorithm:

```
Step 1: Set max to 0 and min to 0.
Step 2: For each number x in the list L, compare it to max.
          Step 2.a: If x is larger, set max to x. Else set min to x
Step 3: Display max and min
Step 4: Stop
```

Algorithm to print numbers from 1 to 20.

Algorithm:

```
Step 1: Initialize X as 0,
Step 2: Increment X by 1,
Step 3: Print X,
Step 4: If X is less than 20 then go back to step 2. Else go to step 5.
Step 5: Stop
```

Algorithm to convert Temperature from Fahrenheit to Celsius

Algorithm:

```
Step 1: Read temperature in Fahrenheit,
Step 2: Calculate temperature with formula C=5/9*(F-32),
Step 3: Print C
Step 4: Stop
```

Algorithm to determine and output whether number N is Even or Odd

Algorithm:

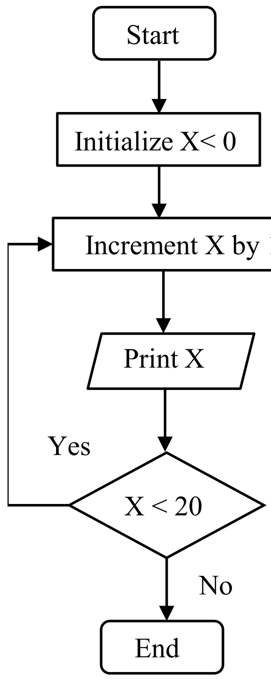
```
Step 1: Read number N
Step 2: Set remainder as N modulo 2
Step 3: If remainder is equal to 0 then number N is even, else number N is odd
Step 4: Print output.
Step 5: Stop
```

Algorithm to determine whether a student passed the Exam or Not

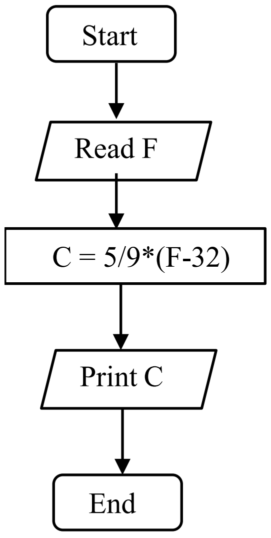
Algorithm:

```
Step 1: Input grades of 4 courses M1, M2, M3 and M4,
Step 2: Calculate the average grade with formula "Grade=(M1+M2+M3+M4)/4"
Step 3: If the average grade is less than 60, print "FAIL", else print "PASS".
Step 4: Stop
```

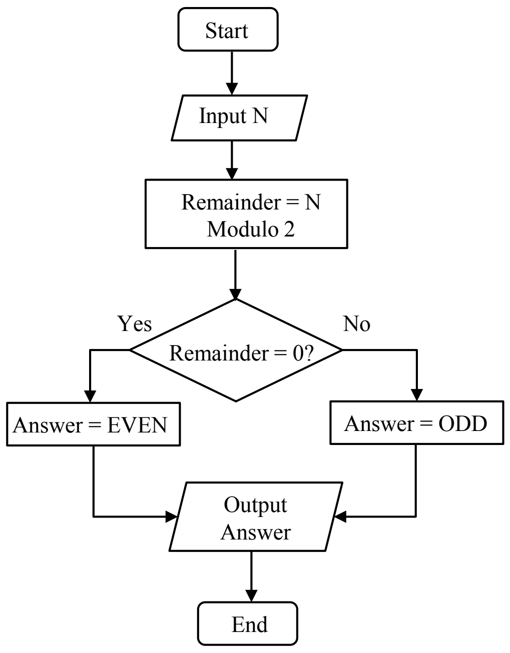
Flowchart to print numbers from 1 to 20



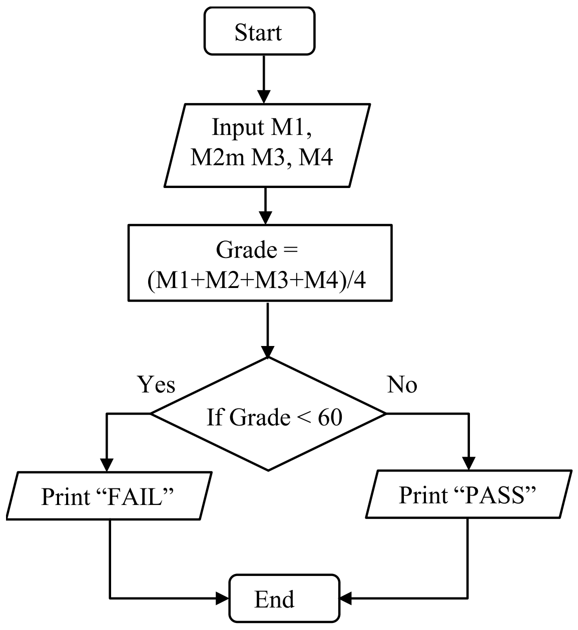
Flowchart to convert Temperature from Fahrenheit to Celsius



Flowchart to determine and output whether number N is Even or Odd



Flowchart to determine whether a student passed the Exam or Not



TWO MARKS QUESTION & ANSWER

1. **What is an algorithm?**

   An algorithm is defined as a step by step procedure for solving a problem. It is a set of rules to solve a problem.

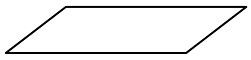
2. **What are the characteristics of an algorithm?**
   - In algorithms each and every instruction should be precise.
   - In algorithms each and every instruction should be unambiguous.
   - The instructions in an algorithm should not be repeated infinitely.
   - Ensure that the algorithm will ultimately terminate.
   - The algorithm should be written in sequence.
   - It looks like normal English.
   - The desired result should be obtained only after the algorithm terminates.

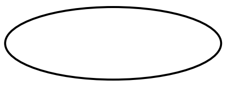
3. **What is a pseudo code?**

   "Pseudo" means imitation of false and "code" refers to the instruction written in the programming language. Pseudo code is programming analysis tool that is used for planning program logic.
   - Pseudo = copy (or) Duplicate, Code = instructions
   - It cannot be compiled or executed
   - It is written in natural language such as English, etc
   - It is used to concentrate on algorithm

4. **What is a flow chart?**

   A Flowchart is a pictorial representation of an algorithm. It is often used by programmer as a program planning tool for organizing a sequence of step necessary to solve a problem by a computer. It contains different symbols.

   Input/Output 

   Terminal 

5. List the ways to represent an algorithm.
- Normal English
- Flowchart
- Pseudo code
- Decision table
- Program

6. Write some rules for drawing a flowchart.

- The standard symbols must be used.
- The arrowheads in the flowchart represent the direction of flow of control in the problem.
- The usual direction of the flow of procedure is from top to bottom or left to right.
- The flow lines should not cross each other.
- Be consistent in using names and variables in the flowchart.
- Keep the flowchart as simple as possible.

7. List few advantages and disadvantages of flowchart.

Advantages:

- Makes logic clear
- Communication
- Effective Analysis
- Useful in Coding
- Proper Testing and Debugging
- Appropriate Documentation

Disadvantages:

- It cannot be prepared for difficult programs
- Alterations and modifications cannot be done easily

8. List the limitations of flowcharts.

- Complex
- Costly
- Difficult to Modify
- No Update

9. List few advantages and disadvantages of pseudo code.

Advantages:

- It can be done easily in any word processor.
- It can be written easily.
- It can be easily modified as compared to flowchart.

Disadvantages:

- It is not visual.

- There are no accepted standards for writing pseudo codes.
- It cannot be compiles nor executed.

10. What is the use of decision box in flowcharts?

The decision symbol is used in a flowchart to indicate the point where a decision is to be made and branching done upon the result of the decision to one or more alternative paths. The criteria for decision making are written in the decision box.

11. What do flowlines show?

Flowlines are solid lines with arrowheads which indicate the flow of operation. They show the exact sequence in which the instructions are to be executed. The normal flow of the flowchart is depicted from top to bottom and left to right.

12. List any two steps involved in problem solving.

The problem solving involves :

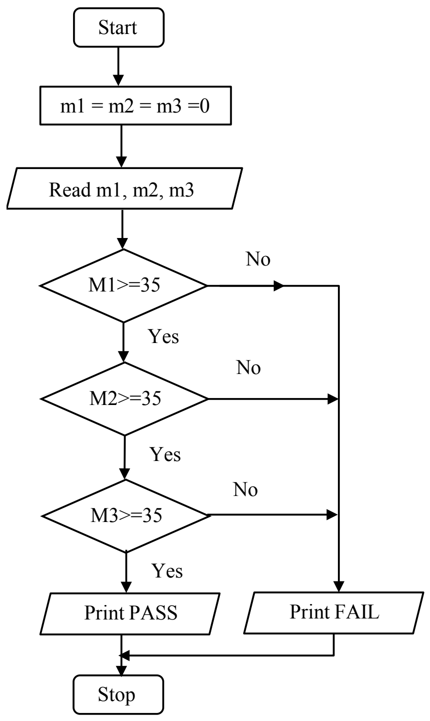
- Detailed study of the problem
- Problem redefinition
- Identification of input data, output requirements and conditions and limitations
- Alternative methods of solution
- Selection of the most suitable method
- Preparation of a list of procedures and steps to obtain the solution
- Generating the output

13. Develop an algorithm and draw the flowchart to get marks for 3 subjects and declare the result. If the marks >= 35 in all the subjects the student passes else fails.

Algorithm:

```
1. Start.
2. Declare three variables m1, m2, m3.
3. Read marks of three subjects m1, m2, m3.
4. If m1 >= 35 goto step 5 else goto step 7
5. If m2 >= 35 goto step 6 else goto step 7
6. If m3 >= 35 print Pass. Goto step 8
7. Print fail
8. Stop
```

Flowchart:

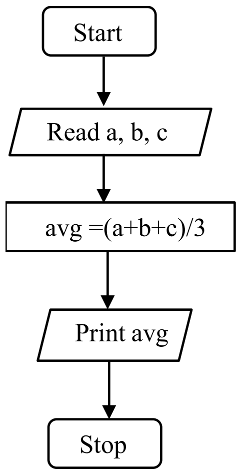


14. Develop an algorithm and draw the flowchart to compute average of three numbers.

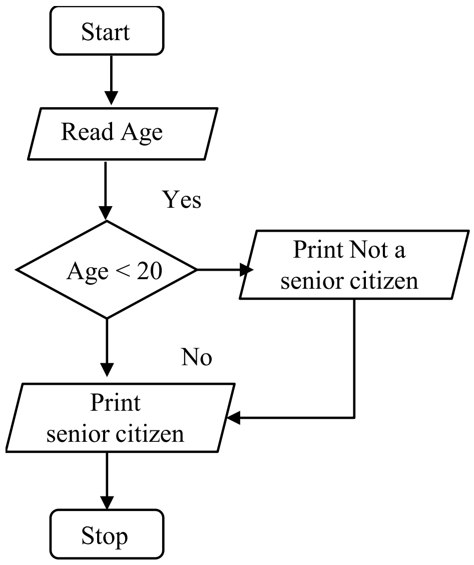
Algorithm:

1. Start
2. Read numbers a,b,c
3. Compute the average as (a+b+c)
4. Print average
5. Stop

Flowchart:



15. Draw the flowchart for following: read age of a person. If age less than 60 then print "Not a senior citizen" otherwise print "Senior Citizen".



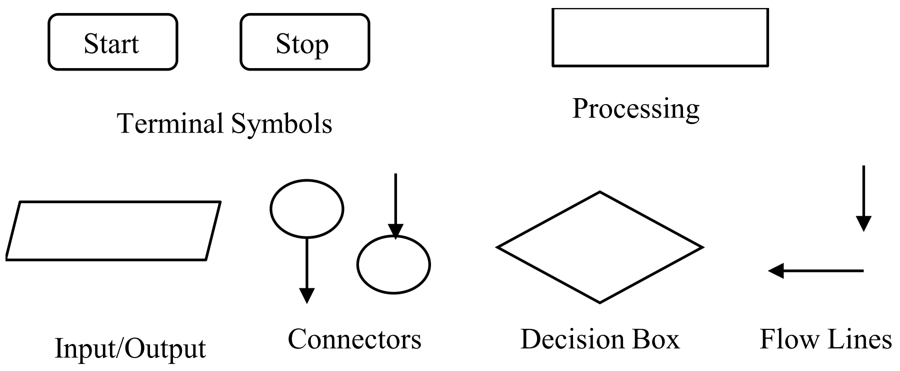
16. Write pseudo code to compute area of rectangle.

```
READ length, breadth
COMPUTE area = length * breadth
DISPLAY area
```

17. Write pseudo code to evaluate student result.

```
READ student's grade
IF student's grade is greater than or equal to 60
PRINT "passed"
ELSE
PRINT "failed"
```

18. Mention the different symbols used in flowchart.



19. What keywords are commonly used in Pseudocode?

Input: READ, INPUT, OBTAIN, GET

Output: PRINT, OUTPUT, DISPLAY, SHOW

Compute: COMPUTE, CALCULATE, DETERMINE

Initialize: SET, INIT

Add one: INCREMENT, BUMP

20. List out the rules in writing the pseudocode.

- Start with an algorithm and phrase it using words that are easily transcribed into computer instructions.
- Indent when you are enclosing instructions within a loop or a conditional clause.
- Avoid words associated with a certain kind of computer language.
- Do not include data declarations in pseudo code.