

1)What is an algorithm?Explain its building blocks with an example? (or)

Explain the control structure of an algorithm with necessary examples

An algorithm is a collection of **well-defined, unambiguous and effectively computable instructions** ,if execute it will return the proper output.

well-defined- The instructions given in an algorithm should be simple and defined well.

Unambiguous- The instructions should be clear,there should not be ambiguity .

effectively computable- The instructions should be written step by step ,which helps computer to understand the control flow .

Algorithm Template Format

The real power of using a template to describe each algorithm is that you can quickly compare and contrast different algorithms and identify commonalities in seemingly different algorithms. Each algorithm is presented using a fixed set of sections that conform to this template.

Name

A descriptive name for the algorithm. It helps to identify the functionalities of an algorithm.For example the algorithm name is **addition of two numbers** ,then it says that the algorithm has been written to add two numbers

Input/Output

Describes the expected format of input data to the algorithm and the resulting values computed.

Context

A description of a problem that illustrates when an algorithm is useful and when it will perform at its best. A description of the properties of the problem/solution that must be addressed and maintained for a successful implementation. They are the things that would cause you to choose this algorithm specifically.

Solution

The algorithm description using real working code with documentation. All code solutions can be found in the associated code repository.

Analysis

The analysis of the algorithm, including performance data and information to help you understand the behavior of the algorithm.

Building Blocks:

It has been proven that any algorithm can be constructed from just three basic building blocks. These three building blocks are **Sequence, Selection, and Iteration(Repetition)**.

Sequence

This describes a sequence of actions that a program carries out one after another, unconditionally.

Execute a list of statements **in order**.

Consider an example,

Algorithm for Addition of two numbers:

Step1: Start
Step 2: Get two numbers as input and store it in to a and b
Step 3: Add the number a & b and store it into c
Step 4: Print c
Step 5: Stop.

The above example is the algorithm to add the two numbers. It says the sequence of steps to be follow to add two numbers. As, Step2 says that should get two numbers from the user and store it in some variable. In step3, the algorithm start doing the process of adding two numbers and step4 print the result generated by step3.

Selection

Selection is the program construct that allows a program to choose between different actions. Choose at most one action from several alternative conditions.

Algorithm to find biggest among 2 nos:

Step1: Start
Step 2: Get two numbers as input and store it in to a and b
Step 3: If a is greater than b then
Step 4: Print a is big
Step 5: else
Step 6: Print b is big
Step 7: Stop

Selection structure executes some set of statements based on the condition given. The above algorithm is an example to find the biggest numbers among two numbers. In Step2, we are getting two numbers as input and storing it in the variables a and b. We are checking whether a is greater than b in step3, if yes then we are printing a is big or we are printing b is big.

Repetition

Repetition (loop) may be defined as a smaller program that can be executed several times in a main program. Repeat a block of statements while a condition is true.

Algorithm to calculate factorial :

Step 1: Start
Step 2: Read the number num.
Step 3: Initialize i is equal to 1 and fact is equal to 1
Step 4: Repeat step 4 through 6 until i is equal to num
Step 5: $fact \leftarrow fact * i$
Step 6: $i \leftarrow i + 1$
Step 7: Print fact
Step 8: Stop

The above algorithm is an example for repetition, where it repeats some process until some condition gets satisfied. Here this algorithm helps to find the factorial of a number. In step 2 we are getting a number as input and store it in num. Step 5 and Step 6 is going to do repeatedly based on the condition given in step 4. Finally, Step 7 prints the result obtained from step 5 and step 6.

With the help of these three building blocks it is easy for us to write an algorithm for any given problem.

2) Explain pseudocode and its control structure with necessary examples.

- “**Pseudo**” means imitation or false and “**code**” refers to the instructions written in a programming language.
- **Pseudocode** is another programming analysis tool that is used for planning a program
- **Pseudocode** is also called Program Design Language(PDL)

Guidelines to write Pseudocode

- Pseudocode is written using structured English.
- Pseudocode should be concise
- Keyword should be in capital letter

Pseudocode is made up of the following logic structure ,

- **Sequential logic**
- **Selection logic**
- **Iteration logic**

Sequence Logic

- It is used to perform instructions in a sequence,that is **one after another**
- Thus,for sequence logic ,pseudocode instructions are written in an order in which they are to be performed.
- The logic flow of pseudocode is from **top to bottom**.

Pseudocode to add two numbers:

```
START  
READ a,b  
COMPUTE c by adding a &b  
PRINT c  
STOP
```

Selection Logic

- It is used for making decisions and for selecting the proper path out of two or more alternative paths in program logic.
- It is also known as decision logic.
- Selection logic is depicted as either an **IF..THEN** or an **IF...THEN..ELSE Structure**.

Pseudocode to add two numbers:

```
START
READ a and b
IF a>b THEN
PRINT "A is big"
ELSE
PRINT "B is big"
ENDIF
STOP
```

Repetition Logic

- It is used to produce loops when one or more instructions may be executed several times depending on some conditions .
- It uses structures called **DO_WHILE, FOR and REPEAT__UNTIL**

Pseudocode to print first 10 natural numbers

```
START
INITIALIZE a←0
WHILE a<10
PRINT a
ENDWHILE
STOP
```

3)What is Flowchart ?List down the symbols and its usage with examples.

A flowchart is a visual representation of the sequence of steps and decision needed to perform a process.


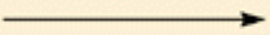


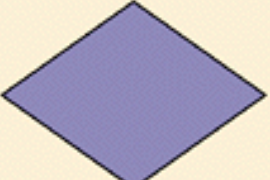
Flowchart for computer programming/algorithms

- As a visual representation of data flow, flowcharts are useful in writing a program or algorithm and explaining it to others or collaborating with them on it.
- You can use a flowchart to spell out the logic behind a program before ever starting to code the automated process.
- It can help to organize big-picture thinking and provide a guide when it comes time to code. More specifically, flowcharts can:

- Demonstrate the way code is organized.
- Visualize the execution of code within a program.
- Show the structure of a website or application.
- Understand how users navigate a website or program.

Flowchart symbols

Here are some of the common flowchart symbols.

Name	Symbol	Use in flowchart
Oval		Denotes the beginning or end of a program.
Flow line		Denotes the direction of logic flow in a program.
Parallelogram		Denotes either an input operation (e.g., INPUT) or an output operation (e.g, PRINT).
Rectangle		Denotes a process to be carried out (e.g., an addition).
Diamond		Denotes a decision (or branch) to be made. The program should continue along one of two routes (e.g., IF/THEN/ELSE).

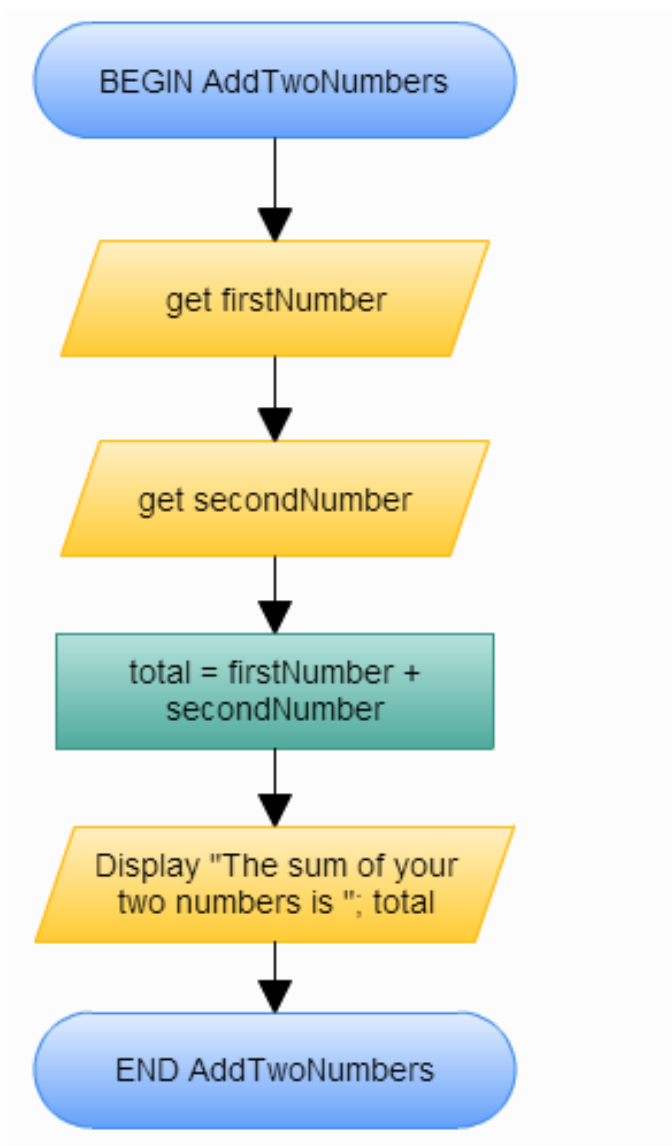
Flowchart is made up of the following logic structure ,

- **Sequential logic**
- **Selection logic**
- **Iteration logic**

Sequence Logic

In a computer program or an algorithm, sequence involves simple steps which are to be executed one after the other. The steps are executed in the same order in which they are written.

Below is an example set of instructions to add two numbers and display the answer.



Selection Logic

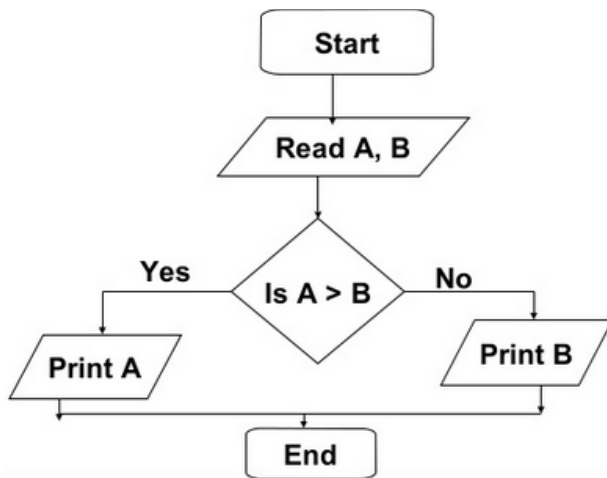
Selection is used in a computer program or algorithm to determine which particular step or set of steps is to be executed. This is also referred to as a 'decision'.

A selection statement can be used to choose a specific path dependent on a condition.

There are two types of selection:

- **binary selection (two possible pathways)**
- **multi-way selection (many possible pathways)**

Following is the example flowchart to find biggest among two numbers



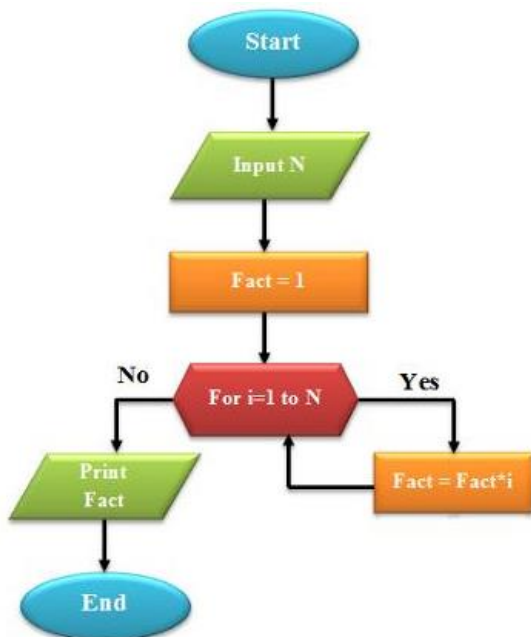
Repetition Logic

Repetition allows for a portion of an algorithm or computer program to be executed any number of times dependent on some condition being met.

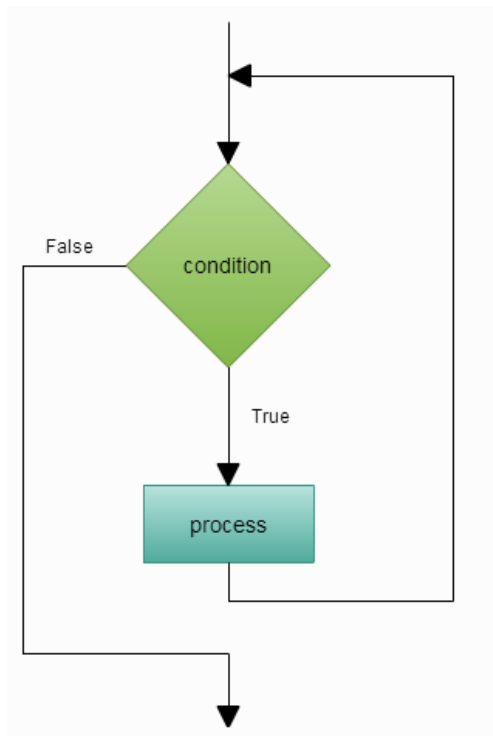
An occurrence of repetition is usually known as a **loop**.

The termination condition can be checked or tested at the **beginning** or **end** of the loop, and is known as a **pre-test** or **post-test**, respectively.

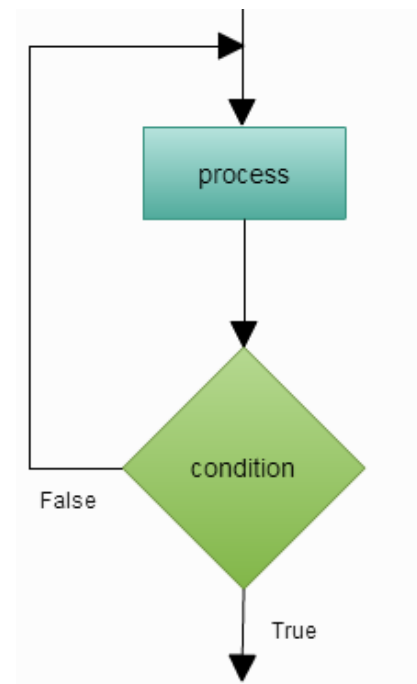
Flowchart to find factorial of given no



pre-test loop



post-test loop



4)What is Programs ?How will you represent an algorithm using programming languages?

- 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.
- To this end, languages 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.

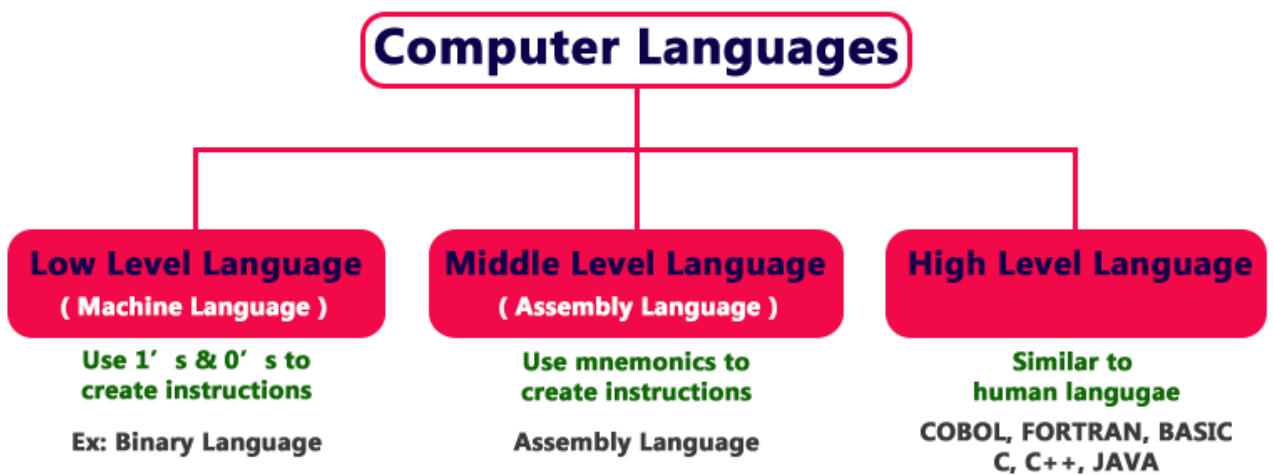
Without an algorithm there can be no program.

- 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.

Simply we can say programming as like below

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.



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.

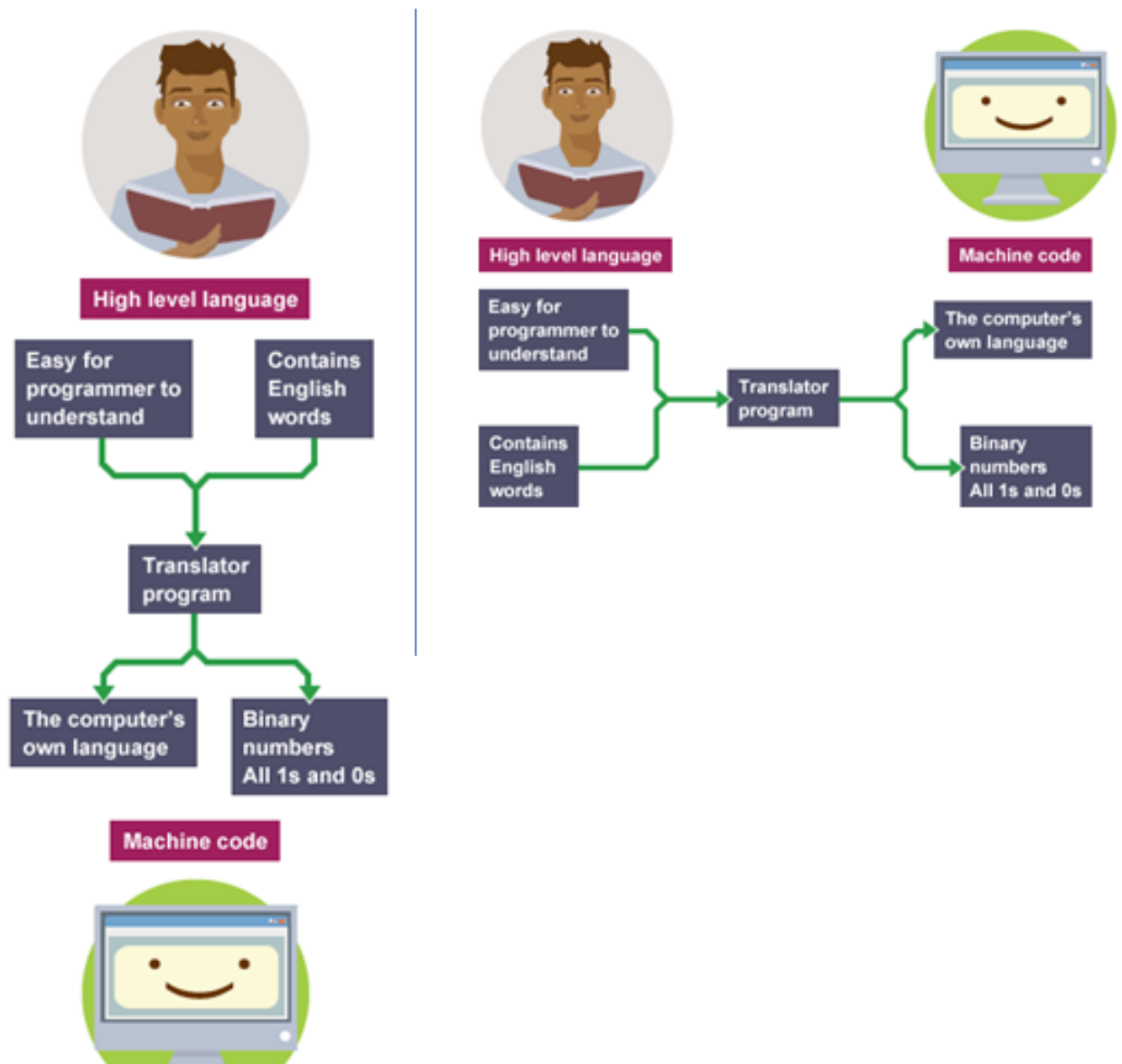
High level Language (Human understandable Language)

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.



5)What is algorithmic problem solving?Explain the process involved in algorithmic development?

“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 .

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.

There are many ways to write an algorithm.

- **Some are very informal.**
- **some are quite formal .**
- **mathematical in nature.**
- **some are quite graphical.**

Once we have an algorithm, we can translate it into a computer program in some programming language. Our algorithm development process consists of five major steps.

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.

When determining the starting point, we should start by seeking answers to the following questions:

- What data are available?
- Where is that data?
- What formulas pertain to the problem?
- What rules exist for working with the data?
- What relationships exist among the data values?

When determining the ending point, we need to describe the characteristics of a solution. In other words, how will we know when we're done? Asking the following questions often helps to determine the ending point.

- What new facts will we have?
- What items will have changed?
- What changes will have been made to those items?
- What things will no longer exist?

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.

High-level algorithm:

Go to a store that sells greeting cards
Select a card
Purchase a card
Mail the card

This algorithm is satisfactory for daily use, but it lacks details that would have to be added were a computer to carry out the solution. These details include answers to questions such as the following.

- "Which store will I visit?"
- "How will I get there: walk, drive, ride my bicycle, take the bus?"
- "What kind of card does Mark like: humorous, sentimental, risqué?"

These kinds of details are considered in the next step of our process.

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.
- Now we need to add details to these steps, but how much detail should we add? Unfortunately, the answer to this question depends on the situation.
- We have to consider who (or what) is going to implement the algorithm and how much that person (or thing) already knows how to do.

If someone is going to purchase Mark's birthday card on my behalf, my instructions have to be adapted to whether or not that person is familiar with the stores in the community and how well the purchaser knows my brother's taste in greeting cards.

- Most of our examples will move from a high-level to a detailed algorithm in a single step, but this is not always reasonable.
- For larger, more complex problems, it is common to go through this process several times, developing intermediate level algorithms as we go.
- Each time, we add more detail to the previous algorithm, stopping when we see no benefit to further refinement.
- This 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.

- First, we need to work through the algorithm step by step to determine whether or not it will solve the original problem.
- Once we are satisfied that the algorithm does provide a solution to the problem, we start to look for other things.
- The following questions are typical of ones that should be asked whenever we review an algorithm.
- Asking these questions and seeking their answers is a good way to develop skills that can be applied to the next problem.

1. Does this algorithm solve a **very specific problem** or does it solve a **more general problem**?
2. If it solves a very specific problem, should it be generalized?
3. Can this algorithm be **simplified**?
4. Is this solution **similar** to the solution to another problem? How are they alike? How are they different?

Problems:

1) Write an algorithm and flowchart to find the minimum number in a list

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

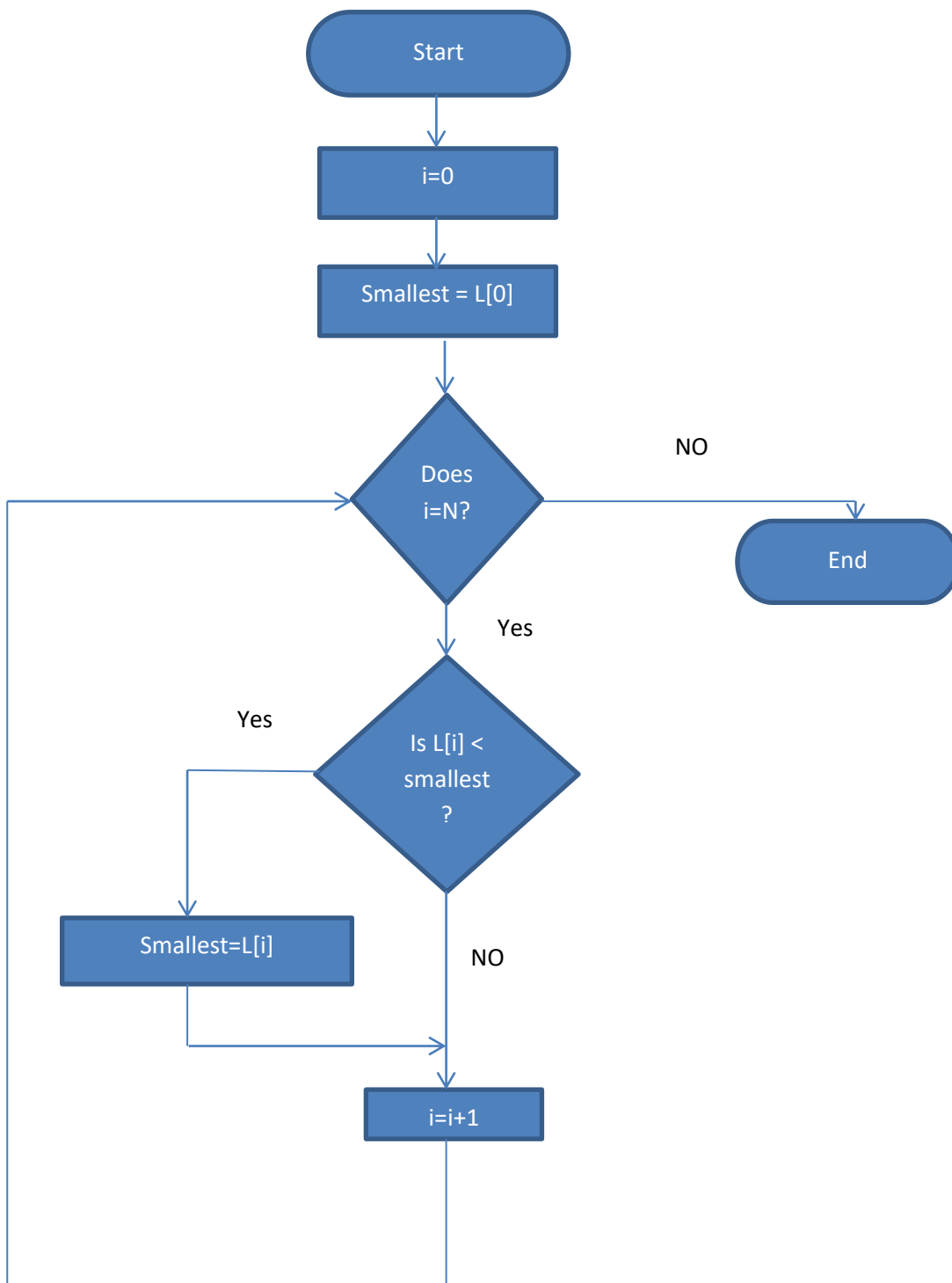
Inputs: A list L of positive numbers. This list must contain at least one number.
(Asking for the smallest number in a list of no numbers is not a meaningful question.)

Outputs: A number n, which will be the smallest number of the 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 and print the result.
6. Stop

Flowchart:



2)Write an algorithm to insert a card in a list of sorted cards.

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

3)Write an algorithm to guess an integer in a range

1. Start
2. Generate a random number from 1 to 20 and store it into the variable number.
3. Ask the user to guess number between 1 and 20 and store it into guess for six chances.
4. Check if guess is equal to number
5. If guess is greater than number print ,the number you guessed is greater
6. If guess is lesser then print,the number you guessed is lesser.
7. If guess is equal to number then,print you guessed is right.
8. If guess is not equal and chance is greater than six print you fail and stop the execution
9. Stop

4)Write an algorithm for tower of Hanoi problem.

1. Start
2. Move disk1 from pegA to pegC
3. Move disk2 from pegA to pegB
4. Move disk3 from pegC to pegB
5. Move disk1 from pegA to pegC
6. Move disk1 from pegB to pegA
7. Move disk2 from pegB to pegC
8. Move disk1 from pegA to peg C
9. Stop

5)Write an algorithm to find thegivennumber is odd or even

Step 1:Start

Step 2: Declare a variable to get a Number

Step 3: Read the input

Step 4: Get the remainder of given number using modulo operator

Step 5: If remainder is 0 prints "Even Number", else print "Odd Number".

Step 6:Stop

6)Write an algorithm to find biggest among 3 numbers

Step 1:Start

Step 1: Declare three integer variables

Step 2: Read the 3 inputs

Step 3: Compare first two numbers and go to Step4

Step 4: If first number is greater than second number then compare first number with third number else go to step 6

Step 5: If first number is greater than third number print first number as biggest number else print third number as biggest

Step 6: Compare second number with third number

Step 7: If second number is greater than third number print second number as biggest number else print third number as biggest

Step8:Stop

7)Write an algorithm to find sum of natural numbers.

```
Step 1:Start
Step 2: Initialize the sum as 0
Step 3: Read the range as input
Step 4: Initialize a counter with 1
Step 5: Overwrite the sum by adding counter value & sum
Step 6: Increment the counter value by 1
Step 7: Repeat the steps 4 & 5 until the counter is less than or equal to range
Step 8: Print the sum
Step 9:Stop
```

8)Write an algorithm to check Armstrong number

```
Step 1: Start
Step 2: Read n
Step 3: n1=n
Step 4: sum=0
Step 5: If n1 is not equal to 0,
rem=n1%10
sum=sum+(rem*rem*rem)
n1=n1/10, goto 5
Step 7: If n==sum, print No. is armstrong
Step 8: Else print No. is not armstrong
Step 9: Stop
```

9)Write an algorithm to find the factorial of given number

```
Step 1:Start
Step 2: Initialize the fact as 1
Step 3: Read the range as input
Step 4: Initialize a counter with 1
Step 5: Overwrite the fact by multiplying counter value & sum
Step 6: Increment the counter value by 1
Step 7: Repeat the steps 4 & 5 until the counter is less than or equal to range
Step 8: Print the fact
Step 9:Stop
```

10) Write an algorithm to generate Fibonacci numbers

Step 1: Start

Step 2: Declare variables l, m, n, sum

Step 3: Initialize the variables, $m=0, n=1$, and $\text{sum} = 0$

Step 4: Enter the number of terms of Fibonacci series to be printed

Step 5: Print First two terms of series

Step 6: Use loop for the following steps

$\text{sum} = m + n;$

$m = n;$

$n = \text{sum};$

Step 7: increase value of i each time by 1

Step 8: print the value of sum

Step 9: Stop