**PROBLEM SOLVING METHODOLOGY**

## DEFINITION-

Problem solving methodology is a process of working through details of a problem to reach a solution . problem solving may include mathematical or systematic operation and can be a gauge of an individual’s critical thinking skill.

## Steps of Problem solving methodology

1. Understand the problem- collect the problem to generate a program.
2. Analyze the problem-analyze the various steps of solving the problem.
3. Design the problem-design the problem by creating flowchart and writing algorithm.
4. Code the program-create coding by using proper programming language.

## ALGORITHM

It is a step by step procedure which defines a set of instruction to be executed in certain order to get the desired output.

## Characteristics of algorithm

1. Precision – the steps are precisely stated(defined).
2. Uniqueness – results of each step are uniquely defined and only depend on the input and the result of the preceding steps.
3. Effectiveness- An algorithm is also generally expected to be effective. This means that all of the operations to be performed in the algorithm must be sufficiently basic that they can in principle be done exactly and in a finite length of time.

Example of not effectiveness :e=1+1/1!+1/2!+....add it to x. Not effective because summation of infinite terms.

1. Finiteness-An algorithm must always terminate after a finite number of steps.
2. Unambiguous- algorithm should be clear and unambiguous. Each steps should be clear and must lead to any one meaning.

Ex: toss a coin . a coin has 2 sides i.e. head and tail. It shows 2 possible values.

## Examples

Write an algorithm to add numbers.

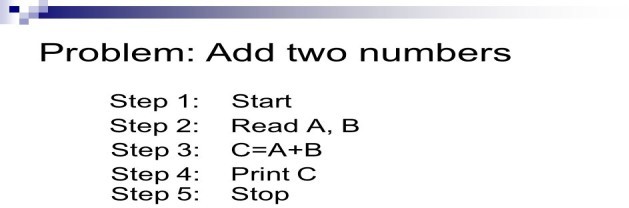
Step1-Read the numbers x,y

Step2-sum=x+y

Step3-print sum

Step-4-end

Or you can write



**Algorithm**

* An algorithm is a step-by-step method for solving some problem.
* An algorithm is a procedure having well defined steps for solving a particular problem.
* Algorithm is finite set of logic or instructions, written in order for accomplish the certain predefined task.
* It is not the complete program or code, it is just a solution (logic) of a problem, which can be represented either as an informal description using a Flowchart or Pseudo code.

**Example:** Design an algorithm to multiply the two numbers x and y and display the result in z.

* Step 1 START
* Step 2 declare three integers x, y & z
* Step 3 define values of x & y
* Step 4 multiply values of x & y
* Step 5 store the output of step 4 in z
* Step 6 print z
* Step 7 end

. Alternatively the algorithm can be written as ?

* Step 1 START MULTIPLY
* Step 2 get values of x & y
* Step 3 z← x \* y
* Step 4 display z
* Step 5 end

### Characteristics of an Algorithm

An algorithm must follow the mentioned below characteristics:

* **Input:** An algorithm must have 0 or well defined inputs.
* **Output:** An algorithm must have 1 or well defined outputs, and should match with the desired output.
* **Feasibility:** An algorithm must be terminated after the finite number of steps.
* **Independent:** An algorithm must have step-by-step directions which is independent of any programming code.
* **Unambiguous:** An algorithm must be unambiguous and clear. Each of their steps and input/outputs must be clear and lead to only one meaning.

# 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).

1. **SEQUENCE**. 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.

Consider an **example-1**, Algorithm for Addition of two numbers:

Step1: Start (you can also write or start your algorithm from step2 and write the step as step1 without writing step1:start)

Step 2: input two numbers a and b Step 3: set sum=a+b

Step 4: Print sum

Step 5: stop // or step5: end

## Example-2

you can write step3 as set a=a+b

set b=a-b set a=a-b

## Example-3

Swapping of two variables using third variable Step1:read the value of a and b

Step2:set temp=a

Set a=b

Set b=temp

Step3:print the value of a and b Step4: end

## Example-4

Algorithm for simple interest.

Step1:read the value of principal, rate and time

Step2:set si=((principal\*rate\*time)/100)

Step3: print si

Step4:end

All these algorithms perform the steps in a purely sequential order.

1. **Decision**

Decision statements are used when the outcome of the process depends on some condition. Decision means choosing between two actions, depending on whether a condition is true or false.

A decision statement can be stated as following manner:-

Using if

If condition then process Example 1:

Algorithm for check whether a person is eligible to vote.

Step 1: read age

Step 2: if age>=18

Step 3: eligible to vote

Step 4: end

* 1. **Using if-else**

**if else** statement is used to add alternative set of **else** statements for **if** condition. In case if condition fails (**FALSE**) then alternatively it will execute **else** statements.

Whenever we have specific requirement like execute set of block statements **if** given condition is TRUE **else** execute other statements in that situation we can use swift **if else** statement.

If condition

then process1/statement1

Else

Process2/statement 2

**Example 1**:algorithm to find whether a number is even or not. Step 1: input a number as a

Step 2:if a%2=0

then print “even”

else

print “odd”

step 3: end

**Example-2:**

write an algorithm to find the larger among two numbers.

Step 1: input two numbers a and b

Step 2: if a>b

then print a is larger

else

print b is larger

step 3: end

* 1. **Using if else-if else**

In, **if else-if else** statement is used to add alternative set of multiple **else-if** and single **else** statements for **if** condition. In case **if** condition fails (FALSE) then alternatively check for another condition in case if it failed for all defined conditions the it will execute **else** statements.

If we want to add multiple condition checks in single program then by using s **if else-if else** statement we can easily add multiple conditions. In **if else-if else** statement we have a chance to add multiple **else if** statements but we are restricted to add only one **if** and **else** conditions in statement.

If condition1

then print staement1

else if condition2

then print statement2

else

print statement3

## example:

Step 1: input two numbers a and b

Step 2: if a>b then

print a is larger

else if a<b then

print b is larger

else

print the numbers are equal

Step 3: end

# 3. Repetition/ Iteration :

**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. ... This construct is often called a guarded loop. Repeat a block of statements while a condition is true.

It can be implemented using construct such as while, do-while and for loop. First we have to initialize the variable.

* + 1. **Using while:**

**while** is used to execute set of statements continuously until, the defined condition is TRUE.

**Q.Write an algorithm to print first 10 natural numbers**

Step 1: Initialize i=0 , n=10

Step 2: repeat step while n<=10

Step 3: Print i

Step 4: end

**Q.Write an 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 factorial←factorial\*i

i←i+1

Step 6: Display factorial Step 7: end

# Using for loop//

using do-while//

# PSEUDOCODE

# pseudocode is an artificial and informal language that helps programmers develop algorithms.pseudocode is a “text-based”detail(algorithmic) design tool.

# It’s simply an implementation of an algorithm in the form of annotations and informative text written in plain English. It has no syntax like any of the programming language and thus can’t be compiled or interpreted by the computer.

# How to write a Pseudo Code?

* **Pseudo code** is a term which is often used in programming and algorithm based fields. It is a methodology that allows the programmer to represent the implementation of an algorithm.
* Simply, we can say that it’s the cooked up representation of an algorithm.
* Often at times, algorithms are represented with the help of pseudo codes as they can be interpreted by programmers no matter what their programming background or knowledge is.
* Pseudo code, as the name suggests, is a false code or a representation of code which can be understood by even a layman with some school level programming knowledge.

### Advantages of Pseudo code

* Improves the readability of any approach. It’s one of the best approaches to start implementation of an algorithm.
* Acts as a bridge between the program and the algorithm or flowchart. Also works as a rough documentation, so the program of one developer can be understood easily when a pseudo code is written out. In industries, the approach of documentation is essential. And that’s where a pseudo-code proves vital.
* The main goal of a pseudo code is to explain what exactly each line of a program should do, hence making the code construction phase easier for the programmer.

### How to write a Pseudo-code?

Arrange the sequence of tasks and write the pseudocode accordingly.

Start with the statement of a pseudo code which establishes the main goal or the aim.

**Example:**

This program will allow the user to check

the number whether it's even or odd.

The way the if-else, for, while loops are indented in a program, indent the statements likewise, as it helps to comprehend the decision control and execution mechanism. They also improve the readability to a great extent.

Use appropriate naming conventions. The human tendency follows the approach to follow what we see. If a programmer goes through a pseudo code, his approach will be the same as per it, so the naming must be simple and distinct.

Use appropriate sentence casings, such as CamelCase for methods, upper case for constants and lower case for variables.

Elaborate everything which is going to happen in the actual code. Don’t make the pseudo code abstract.

Use standard programming structures such as ‘if-then’, ‘for’, ‘while’, ‘cases’ the way we use it in programming.

Check whether all the sections of a pseudo code is complete, finite and clear to understand and comprehend.

Don’t write the pseudo code in a complete programmatic manner. It is necessary to be simple to understand even for a layman or client, hence don’t incorporate too many technical terms.

# STATEMENTS

A statement is defined as an instruction that directs the computer to perform a specific action. In writing pseudo code, we will refer to singular instructions as statements.

When writing pseudo code, we assume that the order of execution of the statements is from top to bottom. This changes when using control structures, functions and exception handling.

## Mathematical operations

Mathematical operations are integral to solution development. They allow us to manipulate the values we have stored.

Here are [common mathematical symbols](https://en.wikipedia.org/wiki/Pseudocode#Common_mathematical_symbols):

**Assignment:** ← or := Example: c ← 2πr, c := 2πr Comparison: =, ≠, <, >, ≤, ≥Arithmetic: +, −, ×, /, mod , Floor/ceiling: ⌊, ⌋, ⌈, ⌉a ← ⌊b⌋ + ⌈c⌉ Logical: and, or Sums, products: Σ Π Example: h ← Σ a ∈A 1/a

**Keywords**:A keyword is a word that is reserved by a program because the word has a special meaning. Keywords can be commands or parameters. Every programming language has its own keywords (reserved words). Keywords **cannot** be used as variable names.

In Pseudocode, they are used to indicate common input-output and processing operations. They are written fully in uppercase.

**START:** This is the start of your pseudocode.

**INPUT:** This is data retrieved from the user through typing or through an input device.

**READ / GET:** This is input used when reading data from a data file.

**PRINT, DISPLAY, SHOW:** This will show your output to a screen or the relevant output device.

**COMPUTE, CALCULATE, DETERMINE**: This is used to calculate the result of an expression.

**SET, INIT:** To initialize values

**INCREMENT, BUMP:** To increase the value of a variable

**DECREMENT:** To reduce the value of a variable

# CONDITIONALS

During algorithm development, we need statements which evaluate expressions and execute instructions depending on whether the expression evaluated to True or False. Here are some common conditions used in Pseudocode:

## IF — ELSE IF — ELSE

This is a conditional that is used to provide statements to be executed if a certain condition is met. This also applies to multiple conditions and different variables.

Here is an if statement with one condition

IF you are happy  
 THEN smile  
ENDIF

Here is an if statement with an else section. Else allows for some statements to be executed if the “if” condition is not met.

IF you are happy THEN  
 smile  
ELSE  
 frown  
ENDIF

We can add additional conditions to execute different statements if met.

IF you are happy THEN  
 smile  
ELSE IF you are sad  
 frown  
ELSE  
 keep face plain  
ENDIF

## CASE

Case structures are used if we want to compare a single variable against several conditions.

INPUT colorCASE color of red: PRINT "red"  
 green: PRINT "green"  
 blue: PRINT "blue"OTHERS  
 PRINT "Please enter a value color"ENDCASE

The OTHERS clause with its statement is optional. Conditions are normally numbers or characters

# ITERATION

To iterate is to repeat a set of instructions in order to generate a sequence of outcomes. We iterate so that we can achieve a certain goal.

## FOR structure

The FOR loop takes a group of elements and runs the code within the loop for each element.

FOR every month in a year Compute number of daysENDFOR

## WHILE structure

Similar to the FOR loop, the while loop is a way to repeat a block of code as long as a predefined condition remains true. Unlike the FOR loop, the while loop evaluates based on how long the condition will remain true.

To avoid a scenario where our while loop runs infinitely, we add an operation to manipulate the value within each iteration. This can be through an increment, decrement, et cetera.

PRECONDITION**:** variable X is equal to 1  
WHILE Population < Limit Compute Population as Population + Births — Deaths

ENDWHILE

# FUNCTIONS

When solving advanced tasks it is necessary to break down the concepts in block of statements in different locations. This is especially true when the statements in question serve a particular purpose. To reuse this code, we create functions. We can then call these functions every-time we need them to run.

Function clear monitor  
 Pass In: nothing  
 Direct the operating system to clear the monitor  
 Pass Out: nothing  
Endfunction

To emulate a function call in pseudocode, we can use the **Call** keyword

call: clear monitor

## Examples of Pseudocode

**create a program to add 2 numbers together and then display the result**.

Start Program  
Enter two numbers, A, B  
Add the numbers together  
Print Sum  
End Program

Compare that pseudocode to an example of a flowchart to add two numbers

|  |
| --- |
| PRINT C  SET C=A+B  ENTER 2 NOS A,B |

**compute the area of a rectangle**:

Get the length, l, and width, w  
Compute the area = l\*w  
Display the area

**compute the perimeter of a rectangle**:

Enter length, l  
Enter width, w  
Compute Perimeter = 2\*l + 2\*w  
Display Perimeter of a rectangle

# CONCLUSION

There are no technical rules for Pseudocode. It is meant to be human readable and still convey meaning and flow.There are different guide and tutorials which lean more towards language-specific pseudocode, examples of such are [Fortran](https://www.fortran.com/) style pseudo code, [Pascal](http://www.pascal-programming.info/index.php) style pseudo code, [C](https://www.cprogramming.com/begin.html) style pseudo code and Structured [Basic](https://en.wikibooks.org/wiki/BASIC_Programming) style pseudo code.

**FLOWCHART**

# A flowchart is simply a graphical representation of steps. It shows steps in sequential order and is widely used in presenting the flow of algorithms, workflow or processes. Typically, a flowchart shows the steps as boxes of various kinds, and their order by connecting them with arrows.

# Guide to Flowchart Symbols, from Basic to Advanced

### 1. The Oval

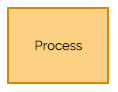
#### An End or a Beginning



The oval, or **terminator,** is used to represent the start and end of a process. Use the Gliffy flowchart tool to drag and drop one of these bad boys and you've got yourself the beginning of a flowchart. Remember to use the same symbol again to show that your flowchart is complete.

### 2. The Rectangle

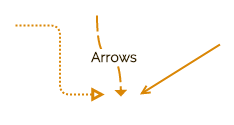
#### A Step in the Flowcharting Process



The rectangle is your go-to symbol once you've started flowcharting. It represents any step in the process you’re diagramming and is the workhorse of the flowchart diagram. Use rectangles to capture **process steps** like basic tasks or actions in your process.

### 3. The Arrow

#### Indicate Directional Flow



The **arrow** is used to guide the viewer along their flowcharting path. And while there are many different types of arrow tips to choose from, we recommend sticking with one or two for your entire flowchart. This keeps your diagram looking clean, but also allows you to emphasize certain steps in your process.

### 4. The Diamond

#### Indicate a Decision

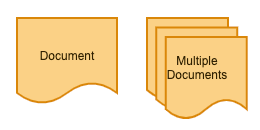


The diamond symbolizes that a **decision** is required to move forward. This could be a binary, this-or-that choice or a more complex decision with multiple choices. Make sure that you capture each possible choice within your diagram.

With those four basic symbols, you likely have everything you need to get started on your own flowchart! Give it a try with Gliffy or read on for more info on intermediate flowcharting symbols.

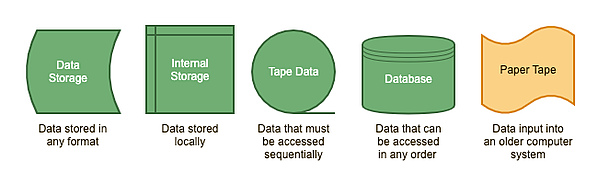
## Intermediate & Advanced Flowchart Symbols

As you know, flowcharts are made up of a sequence of actions, data, services, and/or materials. They illustrate where data is being input and output, where information is being stored, what decisions need to be made, and which people need to be involved. In addition to the basics, these intermediate flowchart symbols will help you describe your process with even more detail.  
**Document Symbols**



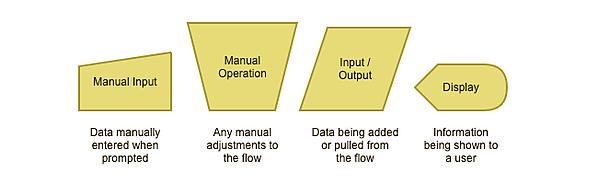
Single and multiple document icons show that there are additional points of reference involved in your flowchart. You might use these to indicate items like “create an invoice” or “review testing paperwork.”

**Data Symbols**



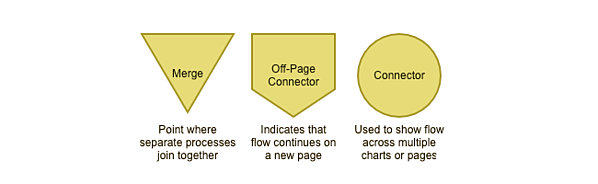
Data symbols clarify where the data your flowchart references is being stored. (You probably won’t use the paper tape symbol, but it definitely came in handy back in the day.)

### Input & Output



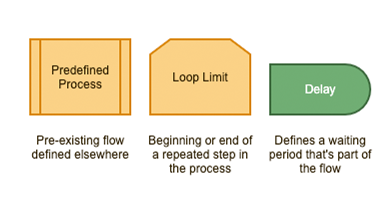
Input and output symbols show where and how data is coming in and out throughout your process.

### Merging & Connecting

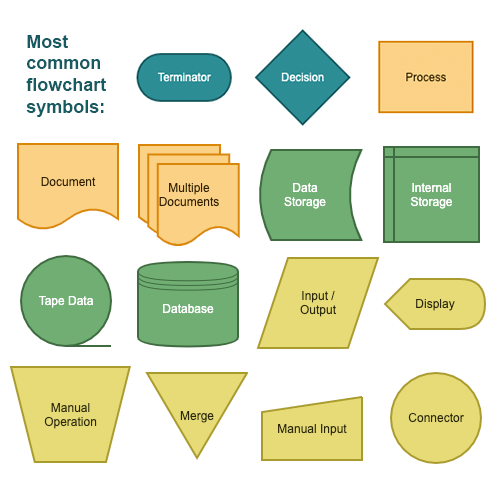


Agreed-upon merging and connector symbols make it easier to connect flowcharts that span multiple pages.

### Additional Useful Shapes



The above are a few additional symbols that prove your flowcharting prowess when put to good use.  
Using Flowchart Symbols



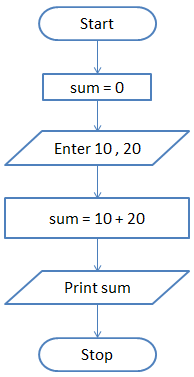
## Types of flowcharts

* **Document Flowcharts:**These “have the purpose of showing existing controls over document-flow through the components of a system. … The chart is read from left to right and documents the flow of documents through the various business units.”
* **Data Flowcharts:**These show “the controls governing data flows in a system. … Data flowcharts are used primarily to show the channels that data is transmitted through the system rather than how controls flow.”
* **System Flowcharts:** These “show the flow of data to and through the major components of a system such as data entry, programs, storage media, processors, and communication networks.”
* **Program Flowcharts:** These show “the controls placed internally to a program within a system.

## ****Example :**** ****Draw a flowchart to input two numbers from user and display the largest of two numbers**** https://media.geeksforgeeks.org/wp-content/uploads/flowchart-3.jpg

## Q1. Add 10 and 20

## Flowchart



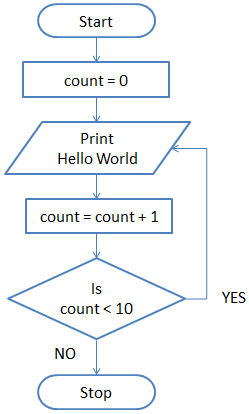
## Q2. Find the sum of 5 numbers

### Flowchart

## flowchart - exercise

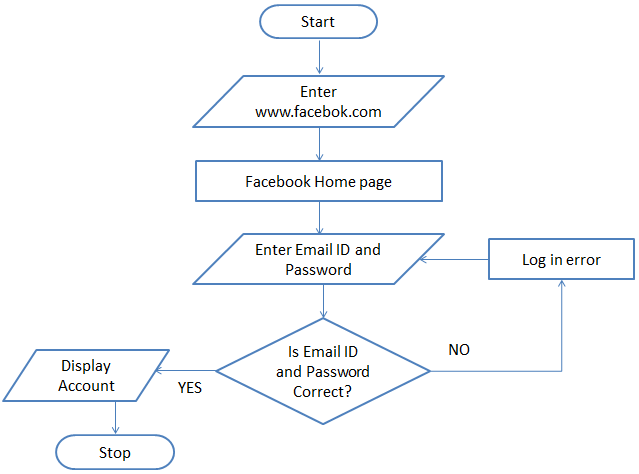
## Q3. Print Hello World 10 times

## Flowchart

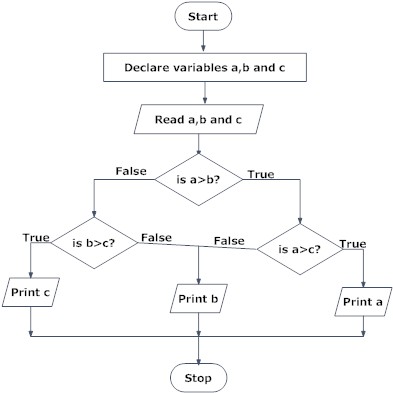


## Q4. Draw a flowchart to log in to facebook account.

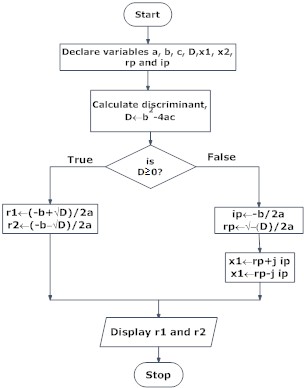
### Flowchart



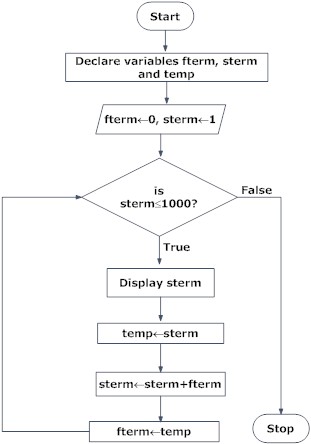
**Draw flowchart to find the largest among three different numbers entered by user.**



**Draw a flowchart to find all the roots of a quadratic equation ax2+bx+c=0**



**Draw a flowchart to find the Fibonacci series till term≤1000.**



**Flowchart for calculate factorial value of a number:**

|  |
| --- |
| [flowchart for calculate factorial value of a number](http://2.bp.blogspot.com/-xDn2DE5BCzQ/UGyfLi6CjWI/AAAAAAAAAio/ACqBsRBWVxI/s1600/factorial.jpg) |

### Difference between Algorithm and Flowchart

|  |  |
| --- | --- |
| **Algorithm** | **Flowchart** |
| It is a procedure for solving problems. | It is a graphic representation of a process. |
| The process is shown in step-by-step instruction. | The process is shown in block-by-block information diagram. |
| It is complex and difficult to understand. | It is intuitive and easy to understand. |
| It is convenient to debug errors. | It is hard to debug errors. |
| The solution is showcased in natural language. | The solution is showcased in pictorial format. |
| It is somewhat easier to solve complex problem. | It is hard to solve complex problem. |
| It costs more time to create an algorithm. | It costs less time to create a flowchart. |

## Generations of programming language

[Programming languages](https://www.includehelp.com/basics/computer-programming-languages.aspx) have been developed over the year in a phased manner. Each phase of developed has made the programming language more user-friendly, easier to use and more powerful. Each phase of improved made in the development of the programming languages can be referred to as a generation. The programming language in terms of their performance reliability and robustness can be grouped into five **different generations**,

1. First generation languages (1GL)
2. Second generation languages (2GL)
3. Third generation languages (3GL)
4. Fourth generation languages (4GL)
5. Fifth generation languages (5GL)

### 1. First Generation Language (Machine language)

The first generation programming language is also called low-level programming language because they were used to program the computer system at a very low level of abstraction. i.e. at the machine level. The machine language also referred to as the native language of the computer system is the first generation programming language. In the machine language, a programmer only deals with a binary number.

**Advantages of first generation language**

* They are translation free and can be directly executed by the computers.
* The programs written in these languages are executed very speedily and efficiently by the CPU of the computer system.
* The programs written in these languages utilize the memory in an efficient manner because it is possible to keep track of each bit of data.

### 2. Second Generation language (Assembly Language)

The second generation programming language also belongs to the category of low-level- programming language. The second generation language comprises assembly languages that use the concept of mnemonics for the writing program. In the assembly language, symbolic names are used to represent the opcode and the operand part of the instruction.

**Advantages of second generation language**

* It is easy to develop understand and modify the program developed in these languages are compared to those developed in the first generation programming language.
* The programs written in these languages are less prone to errors and therefore can be maintained with a great case.

### 3. Third Generation languages (High-Level Languages)

The third generation programming languages were designed to overcome the various limitations of the first and second generation programming languages. The languages of the third and later generation are considered as a high-level language because they enable the programmer to concentrate only on the logic of the programs without considering the internal architecture of the computer system.

**Advantages of third generation programming language**

* It is easy to develop, learn and understand the program.
* As the program written in these languages are less prone to errors they are easy to maintain.
* The program written in these languages can be developed in very less time as compared to the first and second generation language.

**Examples:** FORTRAN, ALGOL, COBOL, C++, C

### 4. Fourth generation language (Very High-level Languages)

The languages of this generation were considered as very high-level programming languages required a lot of time and effort that affected the productivity of a programmer. The fourth generation programming languages were designed and developed to reduce the time, cost and effort needed to develop different types of software applications.

**Advantages of fourth generation languages**

* These programming languages allow the efficient use of data by implementing the various database.
* They require less time, cost and effort to develop different types of software applications.
* The program developed in these languages are highly portable as compared to the programs developed in the languages of other generation.

**Examples:** SOL, CSS, ColdFusion

### 5. Fifth generation language (Artificial Intelligence Language)

The programming languages of this generation mainly focus on constraint programming. The major fields in which the fifth generation programming language are employed are Artificial Intelligence and Artificial Neural Networks

**Advantages of fifth generation languages**

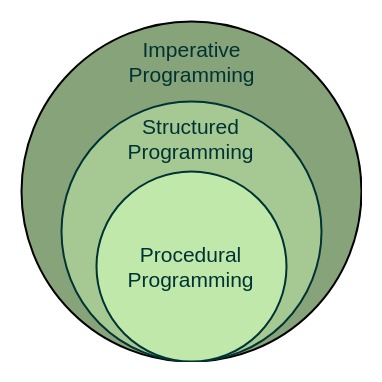
* These languages can be used to query the database in a fast and efficient manner.
* In this generation of language, the user can communicate with the computer system in a simple and an easy manner.

**Examples:** mercury, prolog, OPS5

# Structured Programming Approach with Advantages and Disadvantages

**Structured Programming Approach**, as the word suggests, can be defined as a programming approach in which the program is made as a single structure. It means that the code will execute the instruction by instruction one after the other. It doesn’t support the possibility of jumping from one instruction to some other with the help of any statement like GOTO, etc. Therefore, the instructions in this approach will be executed in a serial and structured manner. The languages that support Structured programming approach are:

* C
* C++
* Java
* C#



On the contrary, in the Assembly languages like Microprocessor 8085, etc, the statements do not get executed in a structured manner. It allows jump statements like GOTO. So the program flow might be random.

## ****Advantages Of Structured Programming Language****

* Structured programming is user-friendly and easy to understand.
* In this programming, programs are easier to read and learn.
* It avoids the increased possibility of data corruption.
* The main advantage of structured programming is reduced complexity.
* Increase the productivity of application program development.
* Application programs are less likely to contain logic errors.
* Errors are more easily found.
* It is easier to maintain.
* It is independent of the machine on which it is used, i.e. programs developed in high-level languages can be run on any computer.

## ****Disadvantages Of Structured Programming Language****

* **Same code repetition**

The code that is written may appear in different parts of the program. It can be vulnerable to various problems because of its location. Programs have variables, which means that they can take on different values at different parts of the program.

* **Lack of encapsulation**

Its lack of availability in structured programming means that programs will be longer.

* **Lack of information hiding**

Information hiding involves isolating design decisions in a computer program that have the greatest chance to change. This protects other parts of the program from modifications if the design decision is changed.