

Basics of a Computer Program- Algorithms, flowcharts (Using Dia Tool), pseudo code.

Basics of a Computer Program:

Irrespective of the programming language you choose to learn, the basic concepts of programming are similar across languages. Some of these concepts include:

- Variable declaration
- Basic syntax
- Data type and structures
- Flow control structures (Conditionals and loops)
- Functional programming
- Object-oriented programming
- Debugging
- IDEs and coding environments

In the next section of this Answer, you will be given a brief introduction to these concepts.

Variable declaration

Variables are containers for storing data values, a memory location for a data type. Variables are created using a declaration or keyword that varies across languages.

Variable names are usually alphanumeric, that is, they contain a-z and 0-9. They can also include special characters like underscore or the dollar sign.

Variables can hold values of any data type supported by the programming language. This value may change during program execution.

Basic syntax

Every programming language has its syntax, and you must learn the fundamental syntax of the language you are learning.

Syntax refers to the set of rules that define the structure of a language. It is almost impossible to read or understand a programming language without its syntax.

For example, let us declare a variable named *greet* and assign the value “Hello World” to it:

Data types and structures

Data types refer to the classification of data. The most common data types include:

- String
- Boolean (true or false)
- Numbers, which includes integers (whole numbers from 1) and floating-point numbers (decimal-base)
- Characters (includes single alphabets or numbers)
- Arrays (a collection of data, usually of the same data type)

A **Data Structure** is a collection of data values. These structures include operations that can be applied to that data. Data structures are important in computer programming for organizing, managing, and storing data quickly and efficiently.

Some common types of data structures include:

- Stacks
- Heaps
- Trees
- Linked lists
- Queues
- Arrays
- Tables
- Graphs

Flow control structures

Flow Control Structures are the fundamental components of computer programs. They are commands that allow a program to “decide” to take one direction or another.

There are three basic types of control structures: sequential, selection, and iteration.

Sequential

The most basic control flow is **sequential control flow**. It involves the execution of code statements one after the other. A real-world example is following a cooking recipe.

Flow chart for sequential control structure

Selection (conditionals)

The basic premise of **selection flow control** is, the computer decides what action to perform based on the result of a test or condition equalling true or false.

Flow chart for selection control structure

Iteration (Loops)

A **loop** is a programming structure that allows a statement or block of code to be run repeatedly until a specified condition is no longer true (will return Boolean, true or false). It is one of the most powerful and fundamental programming concepts.

Flow chart for iteration control structure

Functional programming

Functions are containers that take in a set of inputs and return an output. It is not required for a function to return a value. Pure functions will always give the same result for the same set of inputs.

Functional Programming is a straightforward method of building software that involves using pure functions. This method eliminates the occurrence of data mutation or side effects.

Object-oriented programming

Object-Oriented Programming (OOP) is a programming concept that revolves around 'objects' and 'methods'.

There are four principles of OOP:

- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Debugging

Debugging is a crucial skill. It involves detecting and removing existing and potential errors, defects, or ‘loopholes’ in one’s code.

IDEs and coding environments

IDE stands for **Integrated Development Environment** – they are applications programmers use to write code and organize text groups. It increases a programmer’s efficiency and productivity, and has added features like code completion, code compilation, debugging, syntax highlighting, etc.

Some common examples of IDE’s are:

- Visual Studio code
- IntelliJ IDEA
- NetBeans
- Eclipse

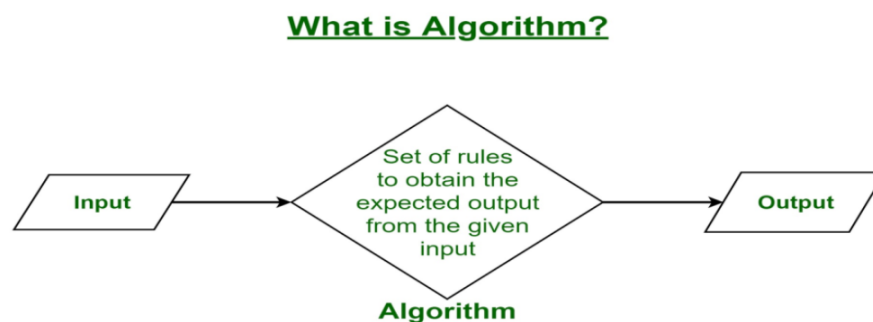
Always remember to write clean, readable codes.

Algorithms

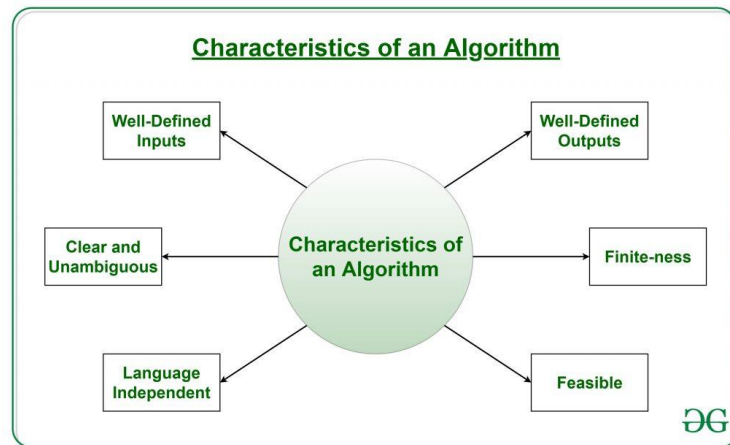
An algorithm is a step by step procedure of solving a problem. The word “algorithm” is derived from the name of the 9th century Persian mathematician Al- Khwarizmi.

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

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



An algorithm must possess the following characteristics:



Well-defined Inputs

- The expected inputs of an algorithm must be well-defined to ensure its correctness, predictability, and repeatability.
- Well-defined inputs ensure that the algorithm's behavior is deterministic, which means, that the same input will always produce the same output.
- Unambiguous inputs help prevent incorrect implementations and misunderstanding of the algorithm's requirements.
- With well-defined inputs, it is easier to optimize the algorithm based on the characteristics of the input.

Well-defined Outputs

- The outputs of an algorithm should be well-defined to ensure that the algorithm produces the intended and accurate result for a given set of inputs.
- It avoids ambiguity and guarantees that the algorithm solves the problem correctly.
- It is also easy to verify the correctness of the algorithm's implementation.
- Well-defined outputs allow you to optimize the algorithm further to achieve the results more efficiently.

Unambiguity

- Ambiguity in the algorithm's description can lead to incorrect implementations and unreliable results. That is why it is important for an

algorithm to be unambiguous.

- It allows the algorithm to be predictable, i.e., the same input produces the same output, which makes debugging an implementation easier.
- It is also easier for unambiguous to be standardized and used widely for different applications.
- While implementing unambiguous algorithms, you can focus more on optimizations rather than handling unexpected errors and edge cases.

Finiteness

- The algorithm should end after a finite amount of time, and it should have a limited number of instructions.
- A finite algorithm ensures that it will eventually stop executing and produce a result.
- An infinite algorithm would never reach a conclusion, which is impractical in real-world scenarios where computation cannot be performed infinitely.
- The time and space complexity can be analyzed in the case of a finite algorithm which is important for performing further optimizations.

Language Independent

- A language-independent algorithm can be easily ported to different programming languages and platforms, making it more adaptable and usable across various environments.
- Being language-independent also makes an algorithm future-proof which means it can be implemented easily using newer programming languages.
- It is important for algorithms to be language-independent in educational settings where students are exposed to different programming languages.
- It also makes it easier to compare the algorithm's performance using different programming languages.

Effectiveness and Feasibility

- An algorithm should be feasible because feasibility indicates that it is practical and capable of being executed within reasonable constraints and resources.
- It also avoids excessive execution times, which can make an algorithm unusable in real-world scenarios.
- Feasible algorithms can be easily implemented using existing hardware infrastructure without using specialized resources.
- They are adopted easily for usage across different fields due to its practical hardware requirements.

Methods for Developing an Algorithm

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

Example: Algorithm for Addition of two numbers:

Step1: Start

Step 2: Get two numbers a and b as input

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

Step 4: Print c

Step 5: Stop.

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

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

Statements: Statement may be a single action in a computer. In a computer statements might include some of the following actions

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

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

The three ways in executing the control flow are

- Sequence
- Selection
- Iteration

Sequence

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

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

Step1: Start

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

Step3: Find the sum of a, b and c

Step4: Divide the sum by 3

Step5: Store the result in variable d

Step6: Print d

Step7: Stop

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

Selection

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

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

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

Step1: Start

Step 2: Get two numbers a and b as input

Step 3: IF $a > b$ THEN

Step 4: PRINT "A is Large"
Step 5: ELSE PRINT "B is Large"
Step 6: ENDIF
Step 7: Stop

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

Iteration

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

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

Step 1: Start
Step 2: Get the value of n.
Step 3: Initialize i=1
Step 4: if ($i \leq n$) go to step 5 else go to step 7
Step 5: Print i value and increment i value by 1
Step 6: Go to step 4
Step 7: Stop

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

The numbers are printed until i becomes greater than n.

Functions

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

Benefits of Using Functions

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

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

Step 1: Start
Step 2: Call the function add ()

Step 3: Stop

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

Subfunction add ()

Step 1: Function start

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

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

Step 4: Print c

Step 5: Return.

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

Pseudo Code

Pseudo –False.

Code- Set of Instructions.

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

Rules for writing Pseudo code

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

Common keywords used in writing a Pseudo code

Comment: //

Start: BEGIN

Stop: END

Input: INPUT, GET, READ

Calculate: COMPUTE, CALCULATE, ADD, SUBTRACT, INITIALIZE

Output: OUTPUT, PRINT, DISPLAY

Selection: IF, ELSE, ENDIF

Iteration: WHILE, ENDWHILE, FOR, ENDFOR

Example: Pseudo code to Add two numbers

BEGIN

GET a, b

ADD c=a+b

PRINT c

END

Advantages:

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

Disadvantages:



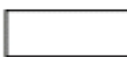





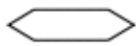
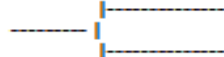
- ☐ It is not visual.
- ☐ We do not get a picture of the design.
- ☐ There is no standardized style or format.
- ☐ For a beginner, it is more difficult to follow the logic or write pseudo code as compared to flowchart

Flow Chart

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

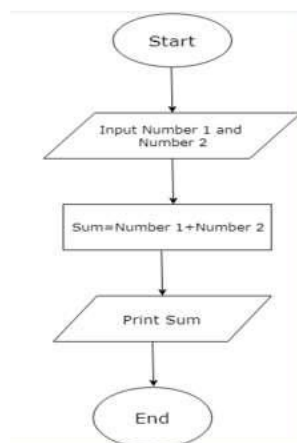
Rules:

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

Symbol Name	Symbol	function
Oval		Used to represent start and end of flowchart
Parallelogram		Used for input and output operation
Rectangle		Processing: Used for arithmetic operations and data-manipulations
Diamond		Decision making. Used to represent the operation in which there are two/three alternatives, true and false etc
Arrows		Flow line Used to indicate the flow of logic by connecting symbols
Circle		Page Connector
		Off Page Connector
		Predefined Process /Function Used to represent a group of statements performing one processing task.
		Preprocessor
		Comments

Advantages:

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



Flowchart for adding two numbers

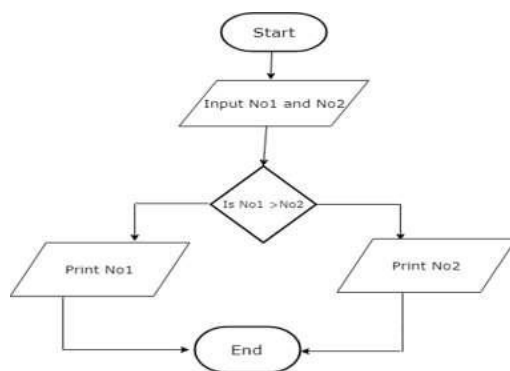
Disadvantages:

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

Example: Add two numbers

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

Example: **Find the largest of two numbers**



Difference between Algorithm, Flowchart and Pseudo code

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