Program
A computer program is a sequence or set of instructions in a programming language for a computer to execute

Programming language
A **programming language** is a computer language programmers use to develop software programs, scripts, or other sets of instructions for computers to execute.
It is a set of instructions written in any specific language (C, C++, Java, Python) to perform a specific task.

Types of programming language

1. Low-level programming language
    i. Machine Language
    ii. Assembly Language
2. High-level programming language
    i. Procedural Oriented programming language
    Example: C, FORTRAN, Basic, Pascal, etc.
    ii. Object-Oriented Programming language
    Example: C++, Java, Python, C#, etc.

Programmer
Alternatively referred to as a **coder** or **software developer**, a **programmer** is an individual who writes code or creates software for a living.

Problem
A problem can be defined as a real-world problem or real-world instance problem for which you need to develop a program or set of instructions. An algorithm is a set of instructions.

Problem Solving
Problem Solving is an essential skill that helps to solve problems in programming. There are specific steps to be carried out to solve problems in computer programming, and the success depends on how correctly and precisely we define a problem. This involves designing, identifying and implementing problems using certain steps to develop a computer.
Basically, they are divided into four categories:

- Analysing the problem
- Developing the algorithm
- Coding
- Testing and debugging

Analysing the Problem

Every problem has a perfect solution; before we are ready to solve a problem, we must look over the question and understand it. When we know the question, it is easy to find the solution for it.By analysing it, we can figure out the outputs and inputs to be carried out. Thus, when we analyse and are ready with the list, it is easy and helps us find the solution easily.

Developing the Algorithm

It is required to decide a solution before writing a program. The procedure of representing the solution in a natural language called an algorithm.It captures and refines all the aspects of the desired solution.

Coding

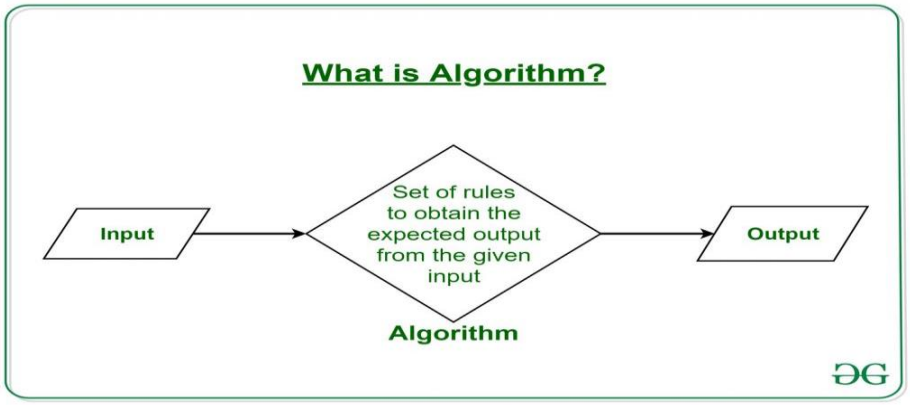
Once we finalise the algorithm, we must convert the decided algorithm into a code or program using a dedicated programming to find a desired solution. In this stage, a wide variety of programming languages are used to convert the algorithm into code.

Testing and Debugging

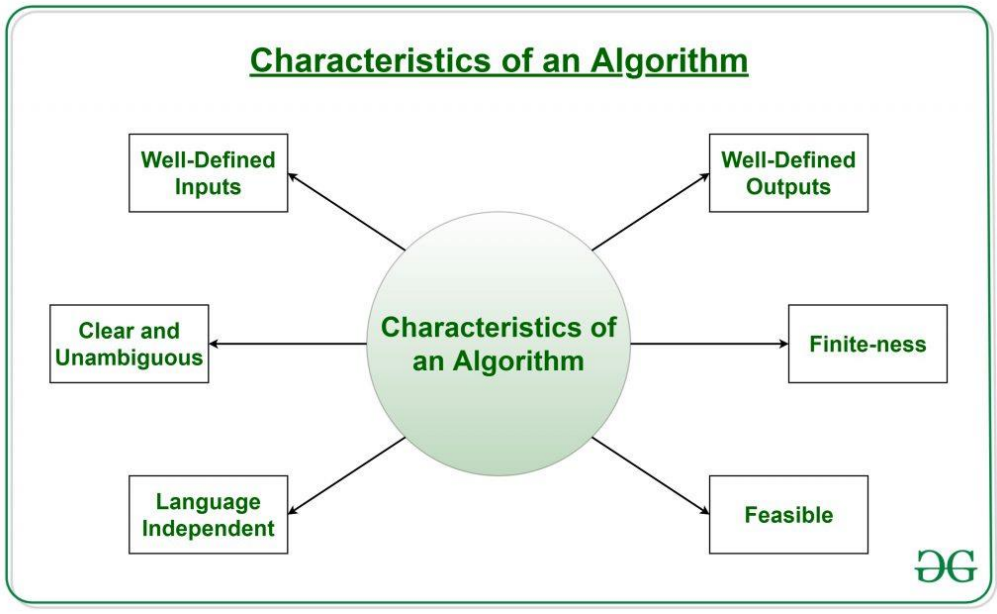
The designed and developed program undergoes several rigorous tests based on various real-time parameters and the program undergoes various levels of simulations. It must meet the user's requirements, which have to respond with the required time. It should generate all expected outputs to all the possible inputs. The program should also undergo bug fixing and all possible exception handling. If it fails to show the possible results, it should be checked for logical errors.

Algorithm

An algorithm is a set of well-defined instructions to solve a particular problem. It takes a set of input(s) and produces the desired output.



What are the Characteristics of an Algorithm?



- **Clear and Unambiguous**: The algorithm should be clear and unambiguous. Each of its steps should be clear in all aspects and must lead to only one meaning.
- **Well-Defined Inputs**: If an algorithm says to take inputs, it should be well-defined inputs. It may or may not take input.
- **Well-Defined Outputs:** The algorithm must clearly define what output will be yielded and it should be well-defined as well. It should produce at least 1 output.
- **Finite-ness:** The algorithm must be finite, i.e. it should terminate after a finite time.
- **Feasible:** The algorithm must be simple, generic, and practical, such that it can be executed with the available resources. It must not contain some future technology or anything.
- **Language Independent:** The Algorithm designed must be language-independent, i.e. it must be just plain instructions that can be implemented in any language, and yet the output will be the same, as expected.

Steps in Algorithm development

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.
It is a very important step in writing an algorithm. Before starting with an algorithm the programmer must obtain maximum information about the problem that needs to be solved.
This step will help the programmer to get a better understanding of the problem which will surely prove to be useful while solving the problem.

Step 2: Analyze the problem.
Proper Analysis of the problem must be done including the data that must be gained, processed and retrieved for generating a valid output.
This step helps the programmer with various processes that must be attained while generating the output along with structure and type of data.

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 a 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

Step 4: Refine the algorithm by adding more detail.
Stepwise refinement is a process for developing a detailed algorithm by gradually adding detail to a high-level algorithm.

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**.

Step 5: Review the algorithm.
The final step is to review the algorithm. What are we looking for? 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.

Properties of Algorithm:

- It should terminate after a finite time.
- It should produce at least one output.
- It should take zero or more input.
- It should be deterministic means giving the same output for the same input case.
- Every step in the algorithm must be effective i.e. every step should do some work.

Advantages of Algorithms:

- It is easy to understand.
- An algorithm is a step-wise representation of a solution to a given problem.
- In Algorithm the problem is broken down into smaller pieces or steps hence, it is easier for the programmer to convert it into an actual program.

Disadvantages of Algorithms:

- Writing an algorithm takes a long time so it is time-consuming.
- Understanding complex logic through algorithms can be very difficult.
- Branching and Looping statements are difficult to show in Algorithms**(imp)**.

Algorithm 1: Add two numbers entered by the 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
```

Algorithm 2: Find the largest number among three numbers

```
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 4: Find the factorial of a number

```
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
```

Algorithm 5: Check whether a number 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 the 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 6: Find the Fibonacci series till the term less than 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
```

Flowcharts

Flowcharts are the graphical representation of the data or the algorithm for a better understanding of the code visually.

It displays step-by-step solutions to a problem, algorithm, or process.

It is a pictorial way of representing steps that are preferred by most beginner-level programmers to understand algorithms of computer science.

A flowchart is a picture of boxes that indicates the process flow in a sequential manner.

Flowchart Symbols

Different flowchart shapes have different conventional meanings. The meanings of some of the more common shapes are as follows:

Terminator

The terminator symbol represents the starting or ending point of the system.

Process

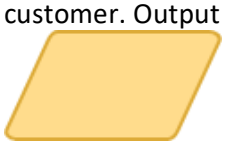
A box indicates some particular operation.

Decision

A diamond represents a decision or branching point. Lines coming out from the diamond indicates different possible situations, leading to different sub-processes.

Input/output

It represents information entering or leaving the system. An input might be an order from a customer. Output can be a product to be delivered.



Connectors

This symbol would contain a letter inside. It indicates that the flow continues on a matching symbol containing the same letter somewhere else on the same page.

Flow

Lines represent the flow of the sequence and direction of a process.

Use of a flowchart

Following are the uses of a flowchart:

- It is a pictorial representation of an algorithm that increases the readability of the program.
- Complex programs can be drawn in a simple way using a flowchart.
- It helps team members get an insight into the process and use this knowledge to collect data, detect problems, develop software, etc.
- A flowchart is a basic step for designing a new process or add extra features.

- Communication with other people becomes easy by drawing flowcharts and sharing them.

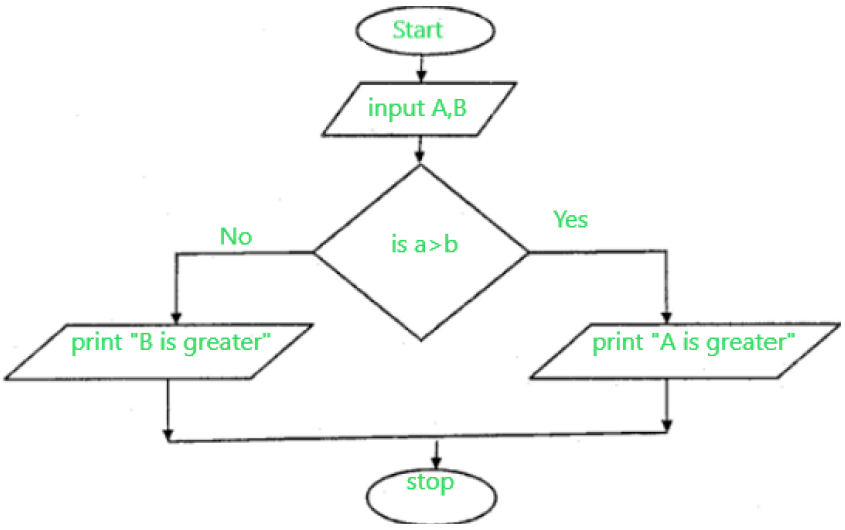
Advantages of Flowchart

- It is the most efficient way of communicating the logic of system.
- It act like a guide for blueprint during program designed.
- It also helps in debugging process.
- Using flowchart we can easily analyze the programs.
- flowcharts are good for documentation.

Disadvantages of Flowchart

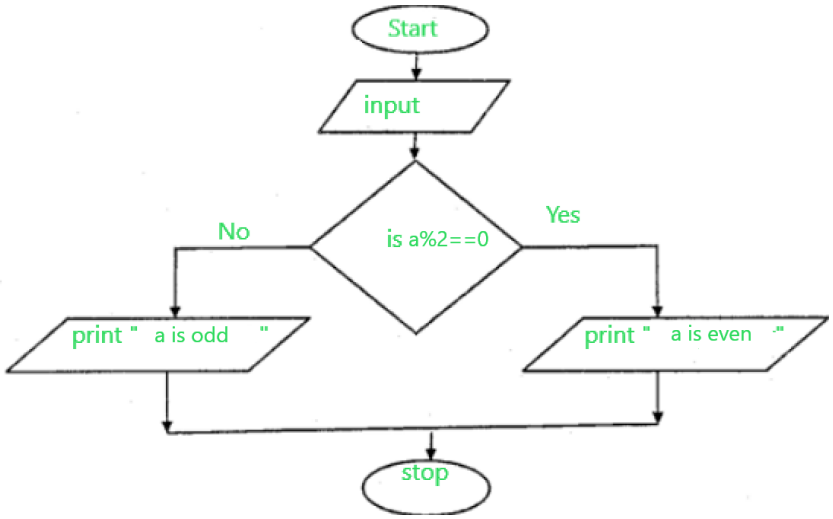
- Flowcharts are difficult to draw for large and complex programs.
- It does not contain the proper amount of details.
- Flowcharts are very difficult to reproduce.
- Flowcharts are very difficult to modify.

Question 1. Draw a flowchart to find the greatest number among the 2 numbers.
Solution:
Algorithm:
1. Start
2. Input 2 variables from user
3. Now check the condition If a > b, goto step 4, else goto step 5.
4. Print a is greater, goto step 6
5. Print b is greater
6. Stop
FlowChart:



Question 2. Draw a flowchart to check whether the input number is odd or even
Solution:
Algorithm:
1. Start
2. Put input a
3. Now check the condition if a % 2 == 0, goto step 5. Else goto step 4
4. Now print("number is odd") and goto step 6
5. Print("number is even")
6. Stop

FlowChart:



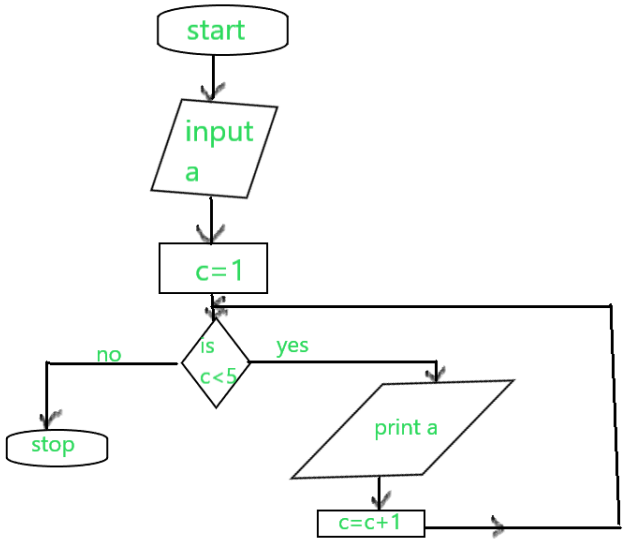
Question 3. Draw a flowchart to print the input number 5 times.

Solution:

Algorithm:

1. Start
2. Input number a
3. Now initialise c = 1
4. Now we check the condition if c <= 5, goto step 5 else, goto step 7.
5. Print a
6. c = c + 1 and goto step 4
7. Stop

FlowChart:



Question 4. Draw a flowchart to print numbers from 1 to 10.

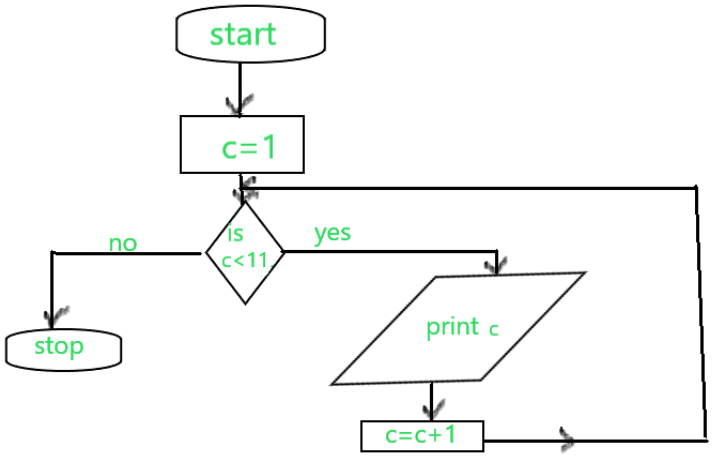
Solution:

Algorithm:

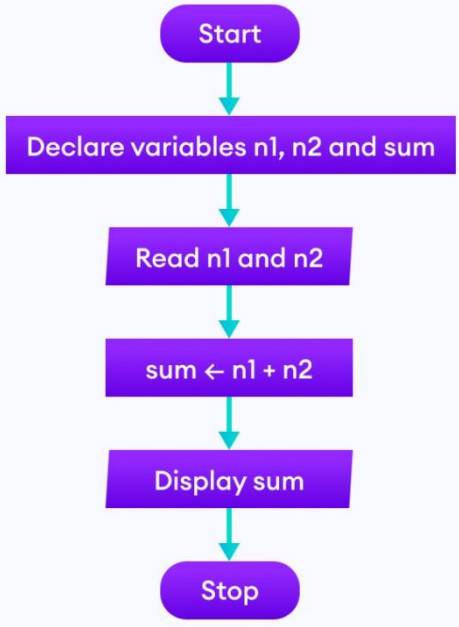
1. Start
2. Now initialise c = 1
3. Now we check the condition if c < 11, then goto step 4 otherwise goto step 6.
4. Print c
5. c = c + 1 then goto step 3

6. Stop

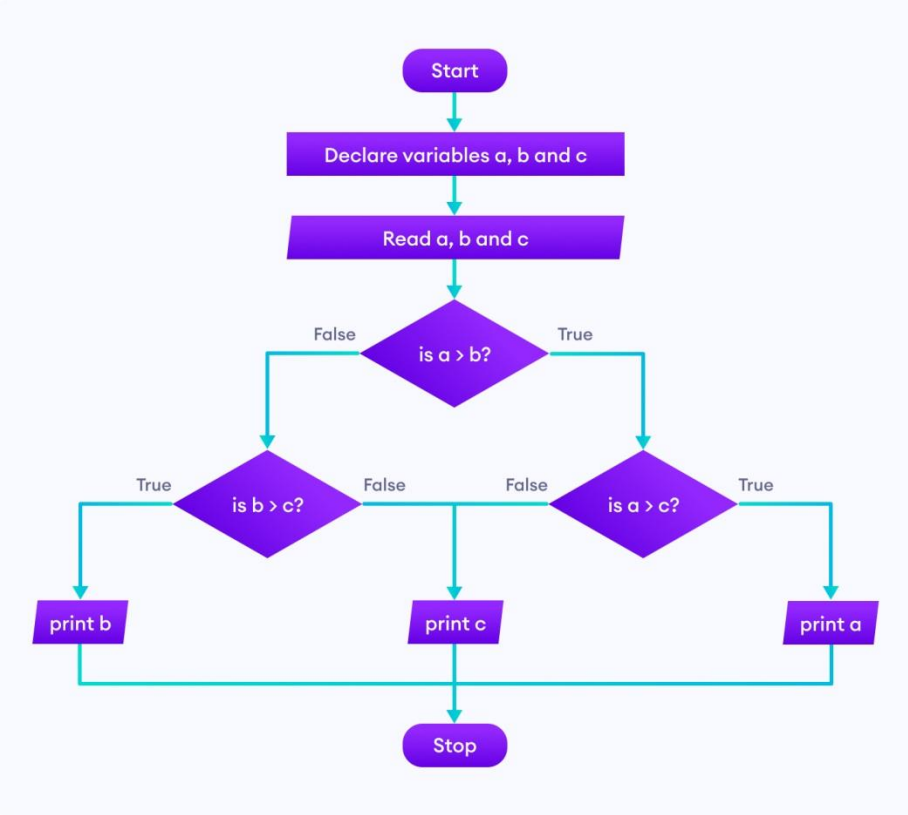
FlowChart:



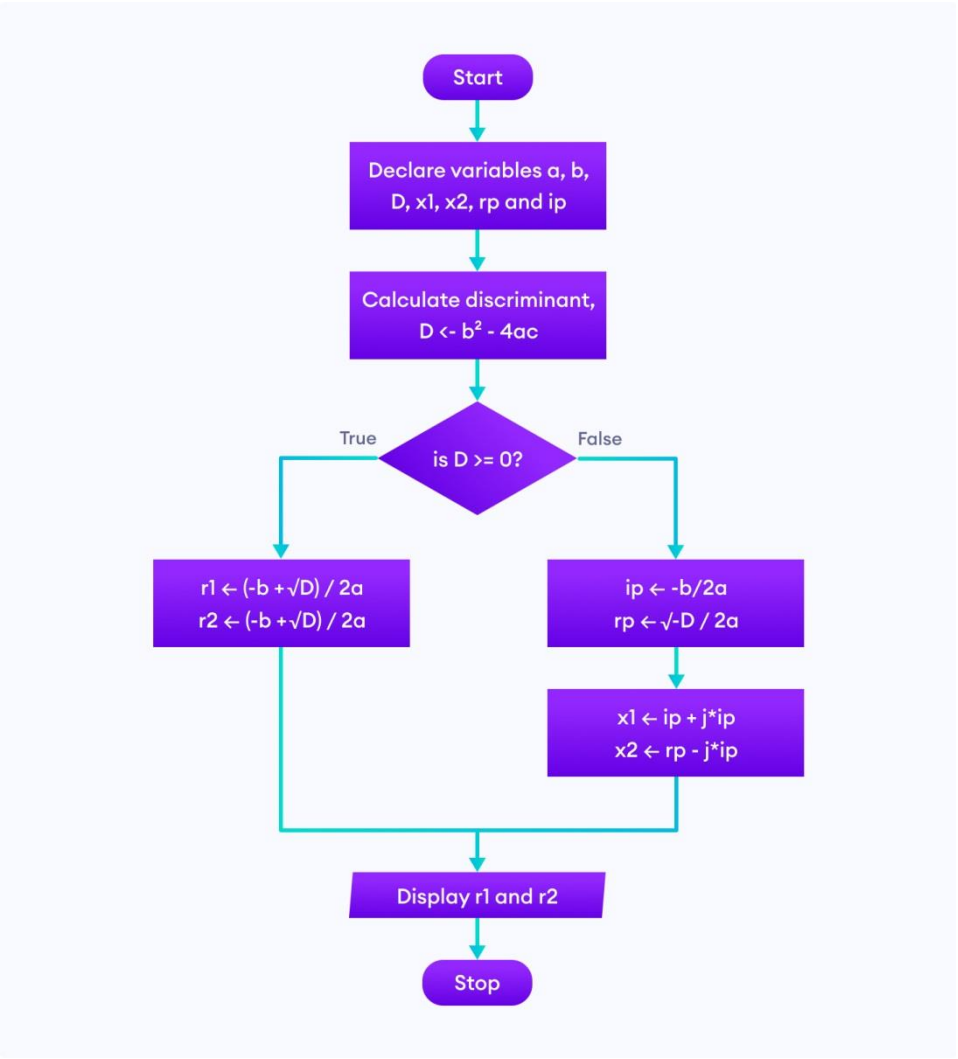
1. Add two numbers entered by the user.



2. Find the largest among three different numbers entered by the user.



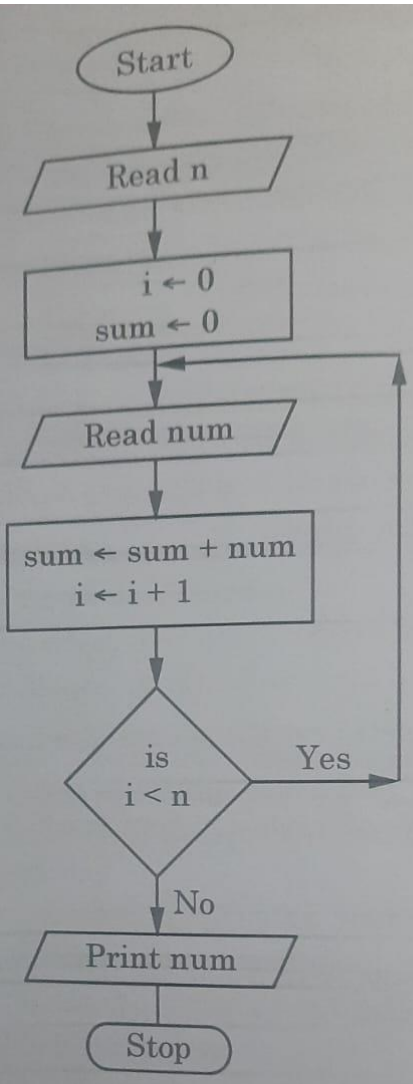
3. Find all the roots of a quadratic equation $ax^2+bx+c=0$


Start
Declare variables a, b, D, x1, x2, rp and ip
Calculate discriminant, D <- b² - 4ac
is D >= 0?
True
False
r1 ← (-b +√D) / 2a
r2 ← (-b +√D) / 2a
ip ← -b/2a
rp ← √-D / 2a
x1 ← ip + j*ip
x2 ← rp - j*ip
Display r1 and r2
Stop

Sum of First n Natural Numbers – Flowchart, Algorithm and Code

Aim : *To find addition*

Flow chart:



Algorithm:

1. Start
2. Print "Enter value of n"
3. Read n
4. Initialise i = 0, sum = 0
5. Read num
6. sum = sum + num
7. i =i + 1
8. If i < n
   Yes-go to 5
9. Print "The sum of given n numbers is", sum
10. Stop