Introduction to Programming Languages

A program is a set of instructions that tells a computer what to do in order to come up with a solution to a particular problem. Programs are written using a programming language.
A programming language is a formal language designed to communicate instructions to a computer. There are two major types of programming languages: low-level languages and high-level languages.

Low-Level Languages

Low-level languages are referred to as 'low' because they are very close to how different hardware elements of a computer actually communicate with each other. Low-level languages are machine oriented and require extensive knowledge of computer hardware and its configuration. There are two categories of low-level languages: machine language and assembly language.

Machine language, or machine code, is the only language that is directly understood by the computer, and it does not need to be translated. All instructions use binary notation and are written as a string of 1s and 0s. A program instruction in machine language may look something like this:

```
10101011100001110100111101
```

However, binary notation is very difficult for humans to understand. This is where assembly languages come in.

An **assembly language** is the first step to improve programming structure and make machine language more readable by humans. An assembly language consists of a set of symbols and letters. A translator is required to translate the assembly language to machine language called the 'assembler.'

While easier than machine code, assembly languages are still pretty difficult to understand. This is why high-level languages have been developed.

High-Level Languages

A high-level language is a programming language that uses English and mathematical symbols, like +, -, % and many others, in its instructions. When using the term 'programming languages,' most people are actually referring to high-level languages. High-level languages are the languages most often used by programmers to write programs. Examples of high-level languages are C++, Fortran, Java and Python.

Learning a high-level language is not unlike learning another human language - you need to learn vocabulary and grammar so you can make sentences. To learn a programming language, you need to learn commands, syntax and logic, which correspond closely to vocabulary and grammar.

The code of most high-level languages is portable and the same code can run on different hardware without modification. Both machine code and assembly languages are hardware specific which means that the machine code used to run a program on one specific computer needs to be modified to run on another computer.

A high-level language cannot be understood directly by a computer, and it needs to be translated into machine code. There are two ways to do this, and they are related to how the program is executed: a high-level language can be compiled or interpreted.

Compiler vs Interpreter

A **compiler** is a computer program that translates a program written in a high-level language to the machine language of a computer.

The high-level program is referred to as 'the source code.' The compiler is used to translate source code into machine code or compiled code. This does not yet use any of the input data. When the compiled code is executed, referred to as 'running the program,' the program processes the input data to produce the desired output.

An **interpreter** is a computer program that directly executes instructions written in a programming language, without requiring them previously to have been compiled into a machine language program.

A little about C++ language

C++ is a cross-platform language that can be used to create high-performance applications. It was developed by **Bjarne Stroustrup**, as an extension to the C language. The language was updated 3 major times in 2011, 2014, and 2017 to C++11, C++14, and C++17.

Why Use C++?

1. C++ is one of the world's most popular programming languages.

2. C++ can be found in today's operating systems, Graphical User Interfaces, and embedded systems.

3. C++ is an object-oriented programming language which gives a clear structure to programs and allows code to be reused, lowering development costs.

4. C++ is portable and can be used to develop applications that can be adapted to multiple platforms.

5. C++ is fun and easy to learn!

6. As C++ is close to C# and Java, it makes it easy for programmers to switch to C++ or vice versa.

How to start writing programs?

Algorithm

Algorithm is a step-by-step procedure, which defines a set of instructions to be executed in a certain order to get the desired output. Algorithms are generally created independent of underlying languages, i.e. an algorithm can be implemented in more than one programming language.

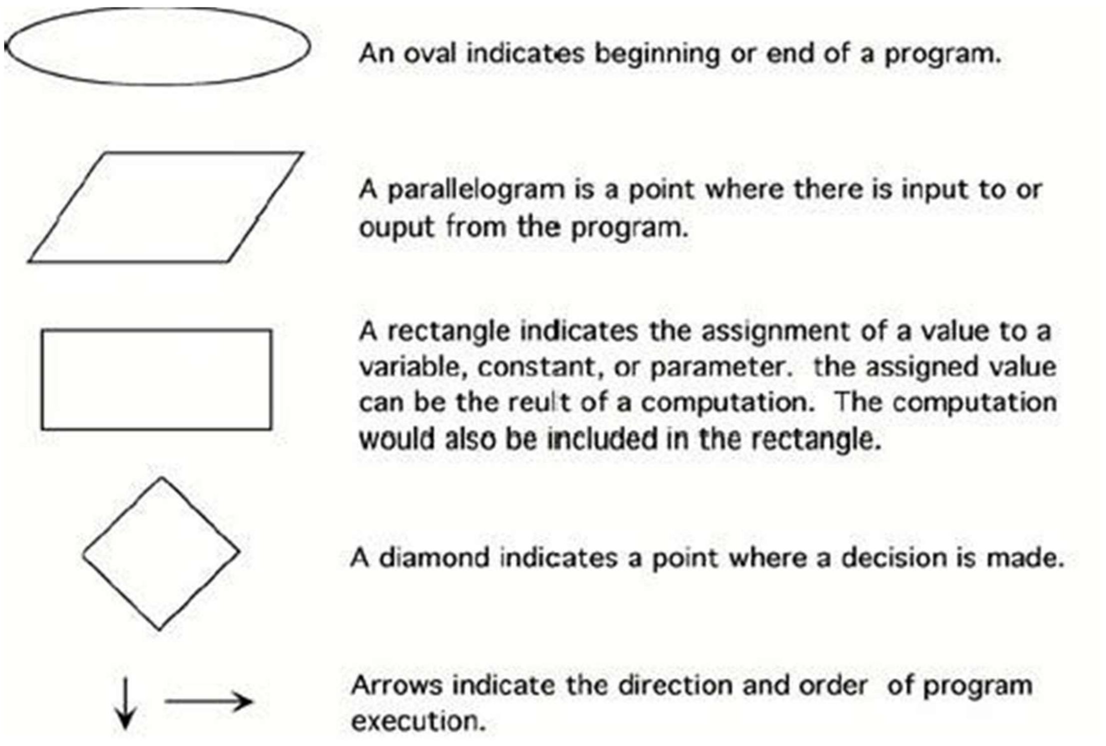
Qualities of a good algorithm

1. Input and output should be defined precisely.
2. Each step in the algorithm should be clear and unambiguous.
3. An algorithm shouldn't include computer code. Instead, the algorithm should be written in such a way that it can be used in different programming languages.

Good, logical programming is developed through good pre-code planning and organization. This is assisted by the use of pseudocode and program flowcharts.

Flowcharts

Flowcharts are written with program flow from the top of a page to the bottom. Each command is placed in a box of the appropriate shape, and arrows are used to direct program flow. The following shapes are often used in flowcharts:



An oval indicates beginning or end of a program.

A parallelogram is a point where there is input to or ouput from the program.

A rectangle indicates the assignment of a value to a variable, constant, or parameter. the assigned value can be the reult of a computation. The computation would also be included in the rectangle.

A diamond indicates a point where a decision is made.

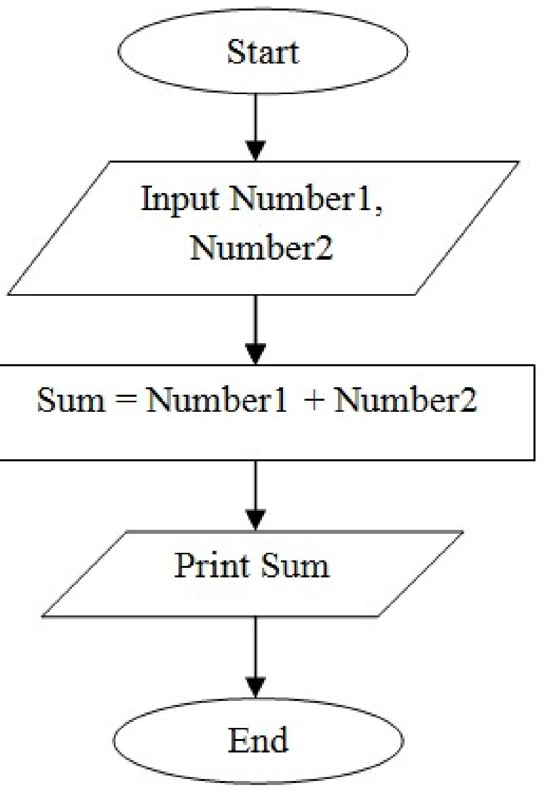
Arrows indicate the direction and order of program execution.

Pseudocode

Pseudocode is a method of describing computer algorithms using a combination of natural language and programming language. It is essentially an intermittent step towards the development of the actual code. It allows the programmer to formulate their thoughts on the organization and sequence of a computer algorithm without the need for actually following the exact coding syntax.

Examples –

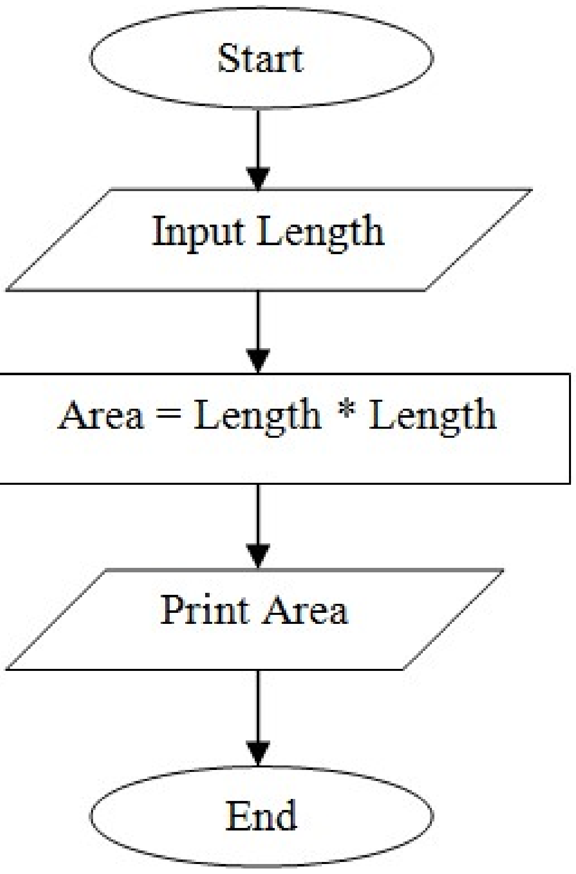
Ques1. Write a flowchart and pseudocode for finding the sum of 2 numbers.



Pseudocode

```
1. Start
2. Input 2 numbers – number1 and number2
3. Add number1 and number2 to find sum
4. Print sum
5. End
```

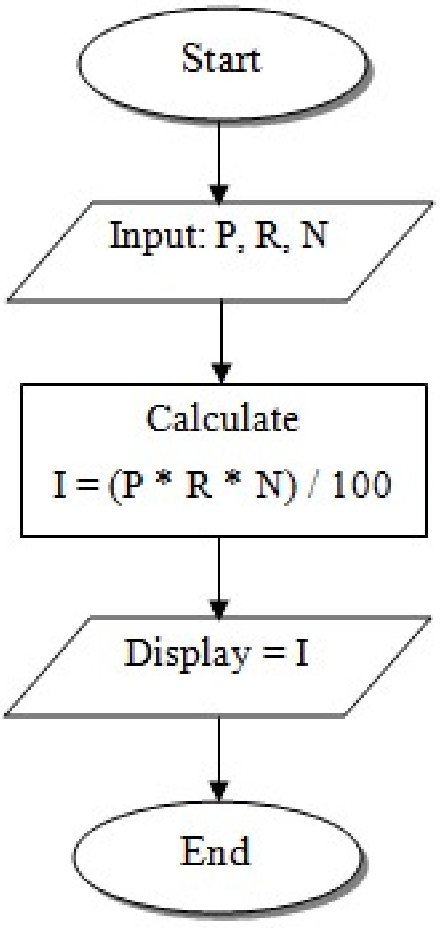
Ques2. Write a flowchart and pseudocode to find the area of a square.



Pseudocode

```
1. Start
2. Input length of the square
3. Calculate area of square as length * length
4. Print area
5. End
```

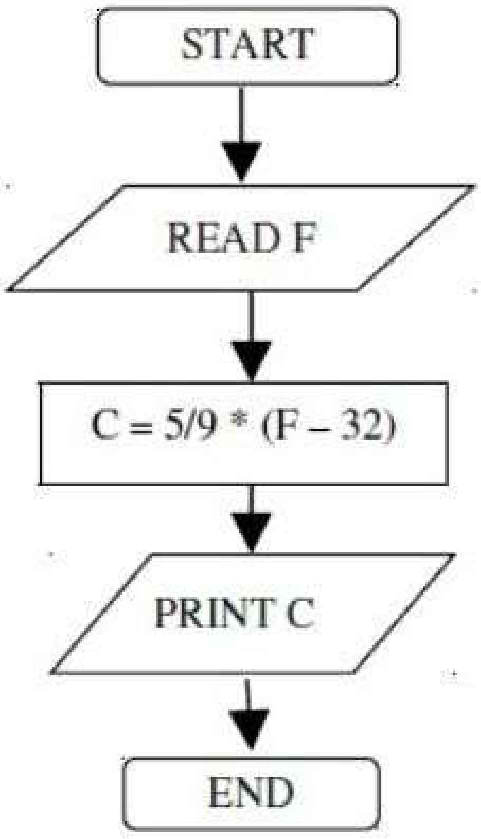
Ques3. Write a flowchart and pseudocode to find simple interest.



Pseudocode

```
1. Start
2. Input a principal as P, rate as R, and time as T
3. Calculate simple interest as P * R * T
4. Print simple interest
5. End
```

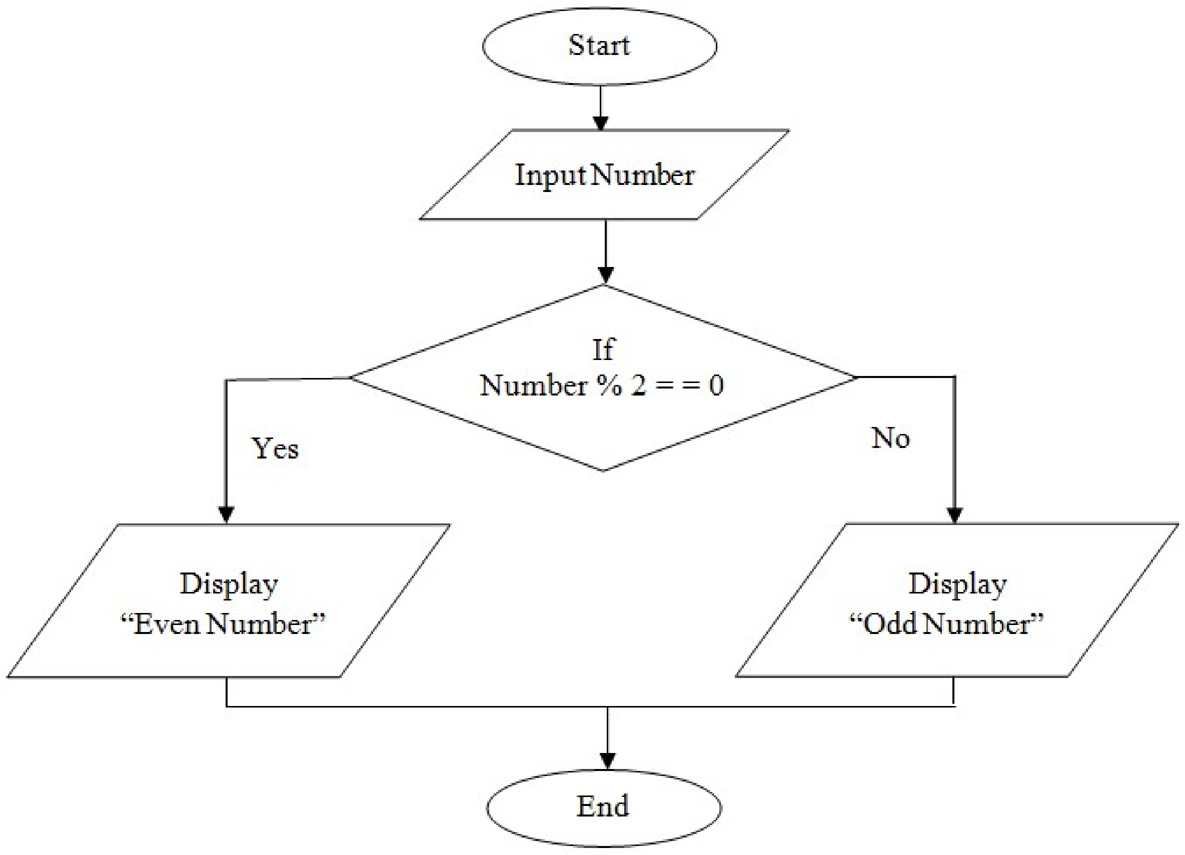
Ques4. Write a flowchart and pseudocode to convert temperature from Fahrenheit (℉) to Celsius (℃).



Pseudocode

```
1. Start
2. Read temperature in Fahrenheit
3. Calculate temperature with formula C=5/9*(F-32)
4. Print C
5. End
```

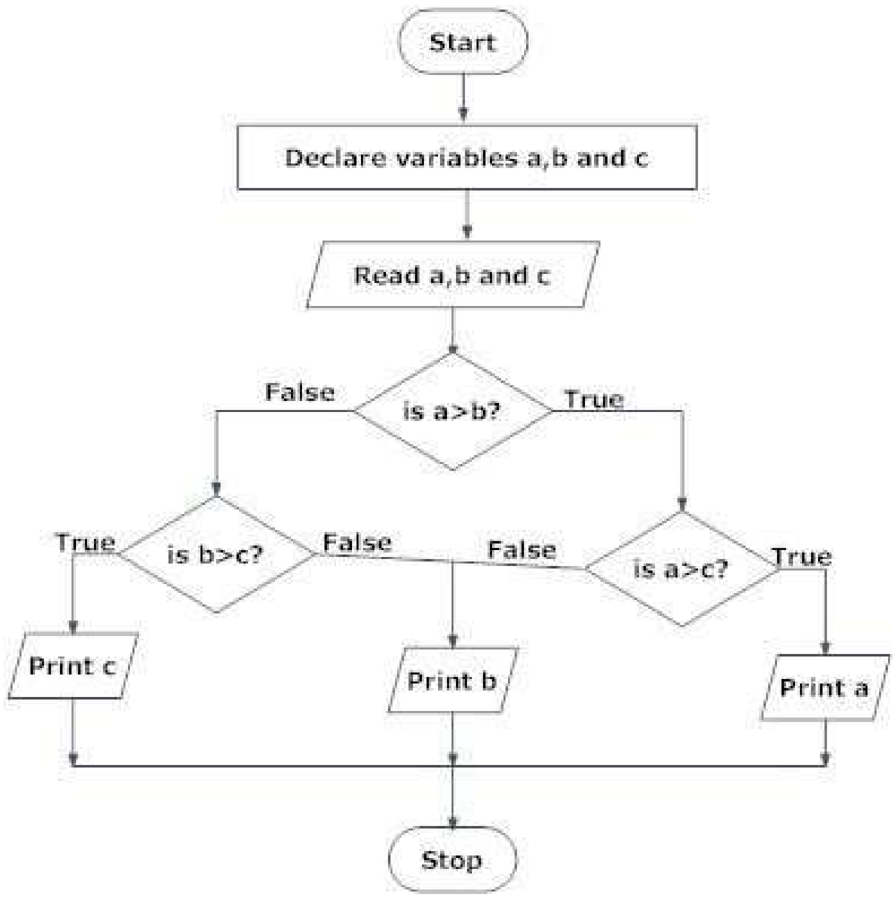
Ques5. Write a flowchart and pseudocode to check if a number is odd or even.



Pseudocode

1. Start
2. Input a number
3. If remainder when number is divided is 0, print "Even number"
4. Else print "Even number"
5. End

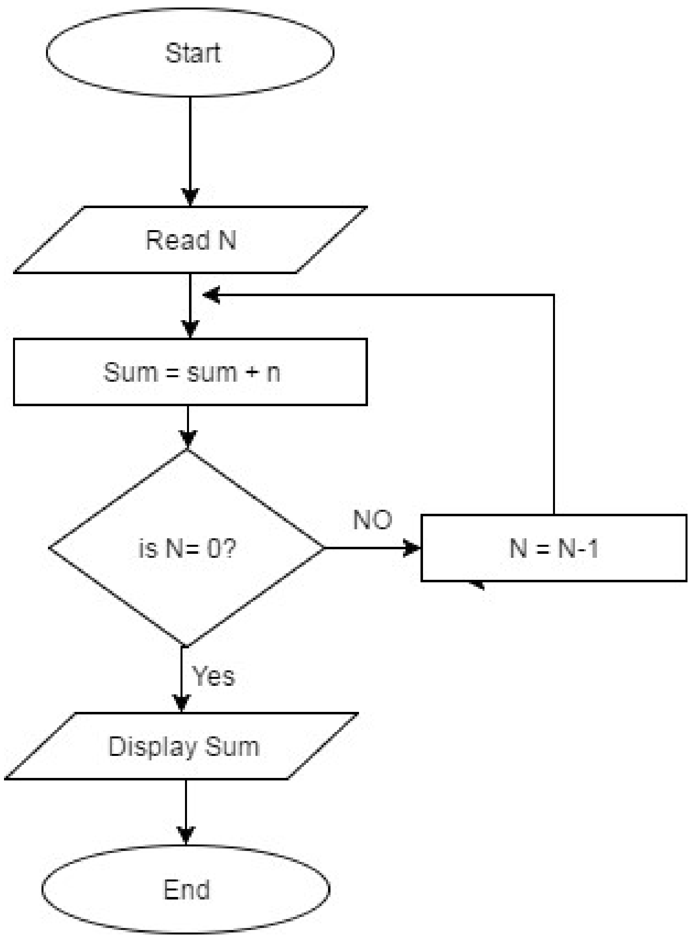
Ques6. Write a flowchart and pseudocode to find the max of 3 numbers.



Pseudocode

```
1. Start
2. Read three numbers – a,b and c
3. Compare a and b, if a > b
           Compare a and c, if a > c
                   Print a
           Else
                   Print c
   Else
          Compare b and c, if b > c
                   Print b
           Else
                   Print c
4. End
```

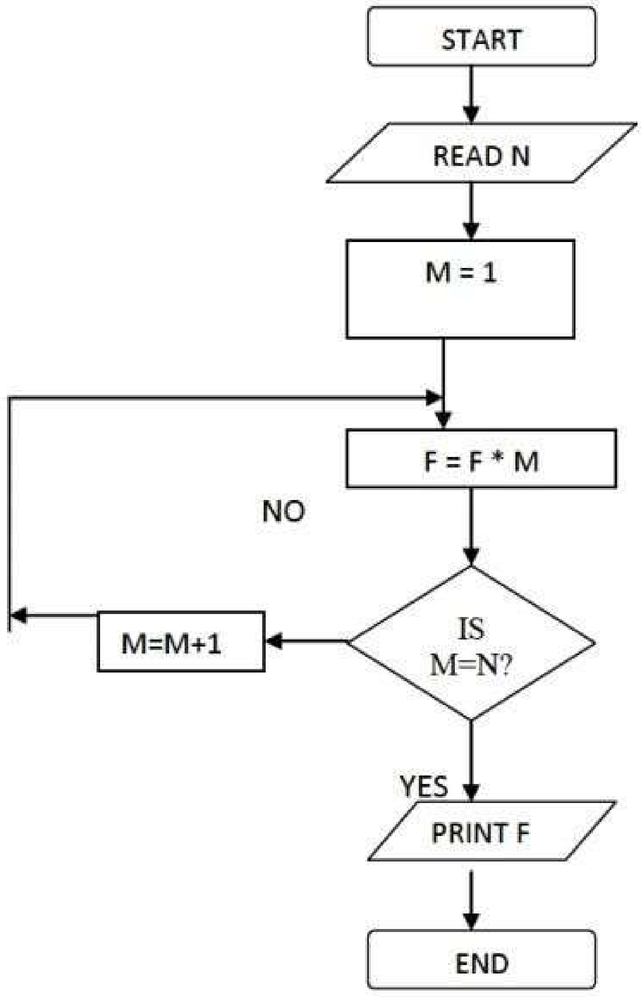
Ques7. Write a flowchart and pseudocode to find the sum of n natural numbers.



Pseudocode

```
1. Start
2. Read a number n
3. Repeat the following till n becomes equal to 0
       Add n in sum
       Decrement n
4. Print sum
5. End
```

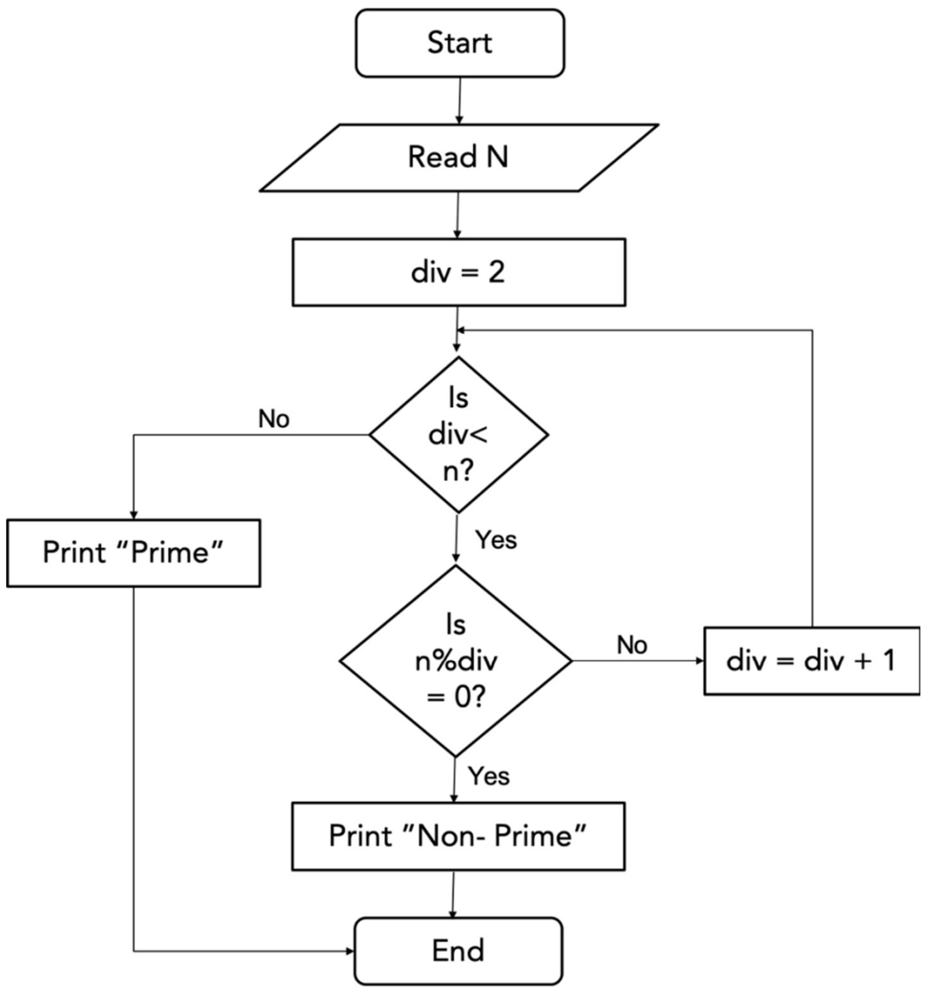
Ques8. Write a flowchart and pseudocode to find the factorial of a number.



Pseudocode

```
1. Start
2. Read a number n
3. Initialise m =1
4. Repeat the following till m becomes equal to n
      Multiply m with factorial
      Increment m
5. Print factorial
6. End
```

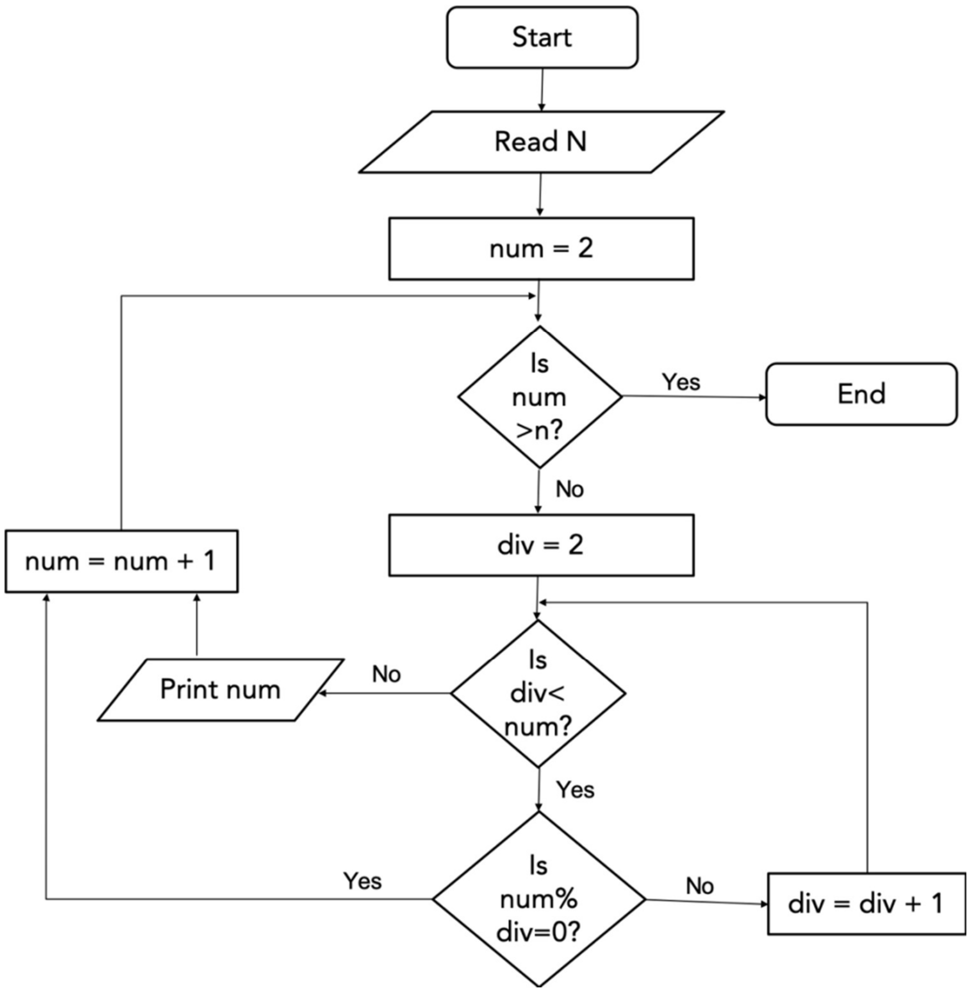
Ques9. Write a flowchart and pseudocode to find if a number is prime or not.



Pseudocode

```
1. Start
2. Read a number n
3. Initialise divisor = 2
4. Repeat till divisor < n
      If n is divisible by divisor
          Print “Non-Prime”
          End
      Else
          divisor = divisor + 1
5. Print “Prime”
6. End
```

Ques10. Write a flowchart and pseudocode to print all prime numbers till n.



Pseudocode

```
1. Start
2. Read a number n
3. Initialise num = 2
4. While num <= n do
       Initialise div = 2
       While div < num do
           If n is divisible by div
             num = num + 1
          Else
             div = div + 1
       print num
       num = num + 1
5. End
```