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.

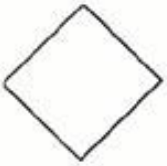
Flowcharts

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.

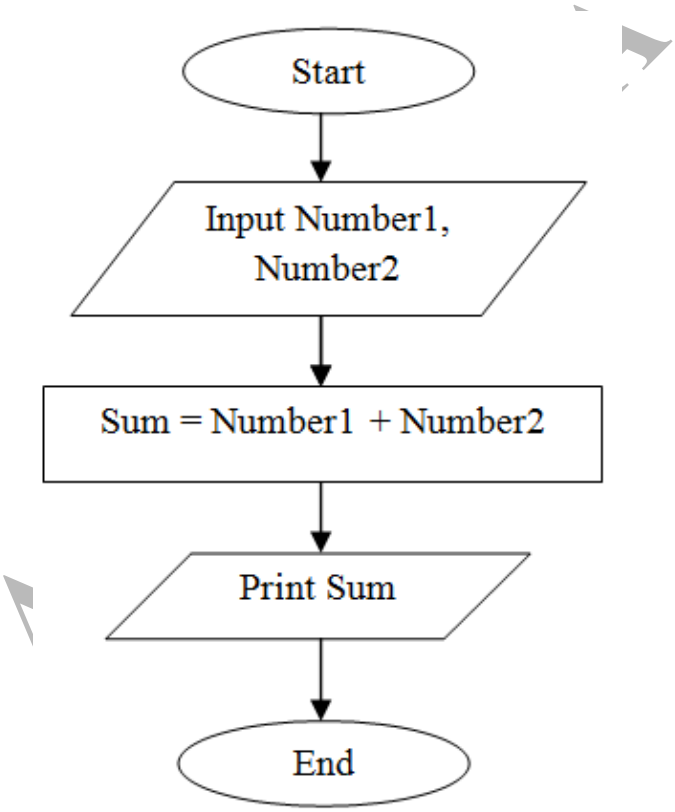
Arrows indicate the direction and order of program execution.

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 –

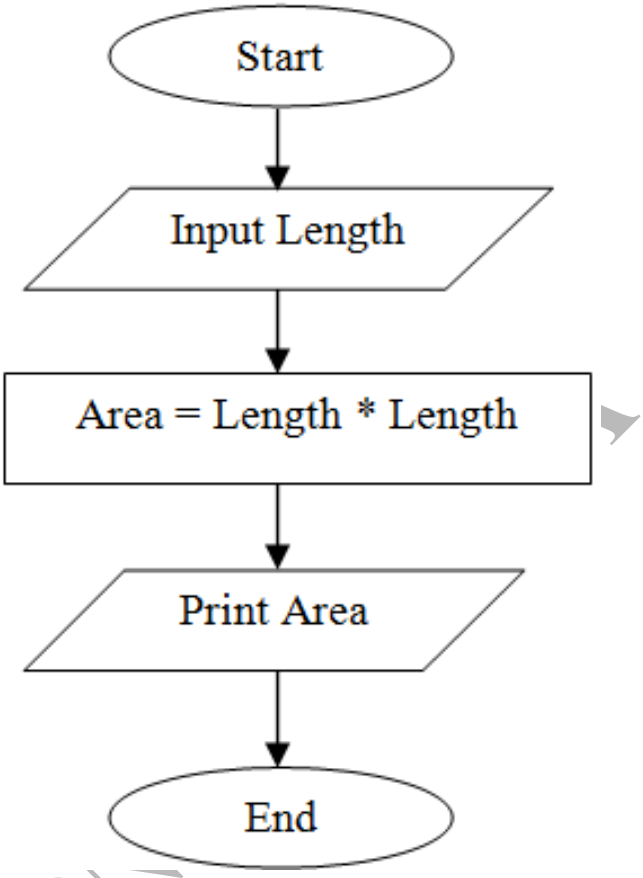
Ques1. Write a flowchart and pseudocode for finding the sum of 2 numbers.



Pseudocode

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

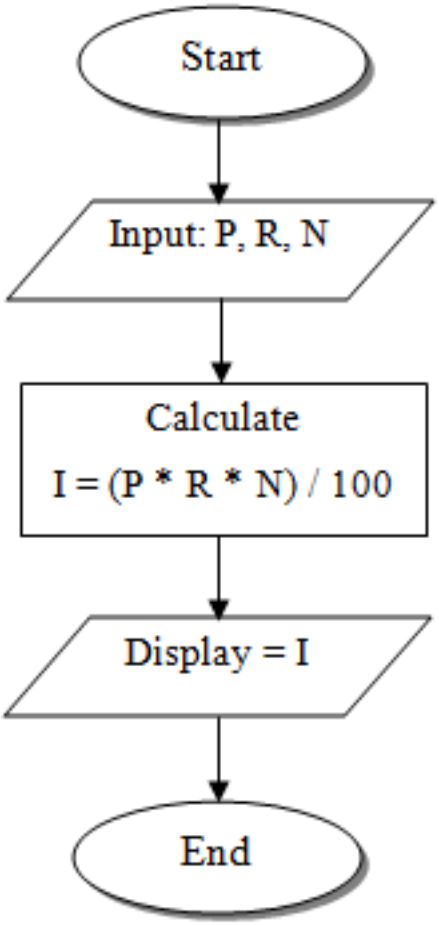
Ques2. Write a flowchart and pseudocode to find the area of a square.



Pseudocode

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

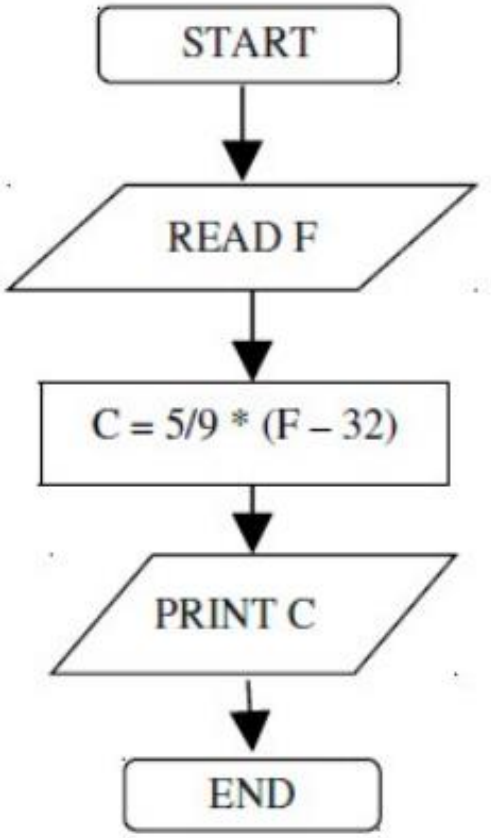
Ques3. Write a flowchart and pseudocode to find simple interest.



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

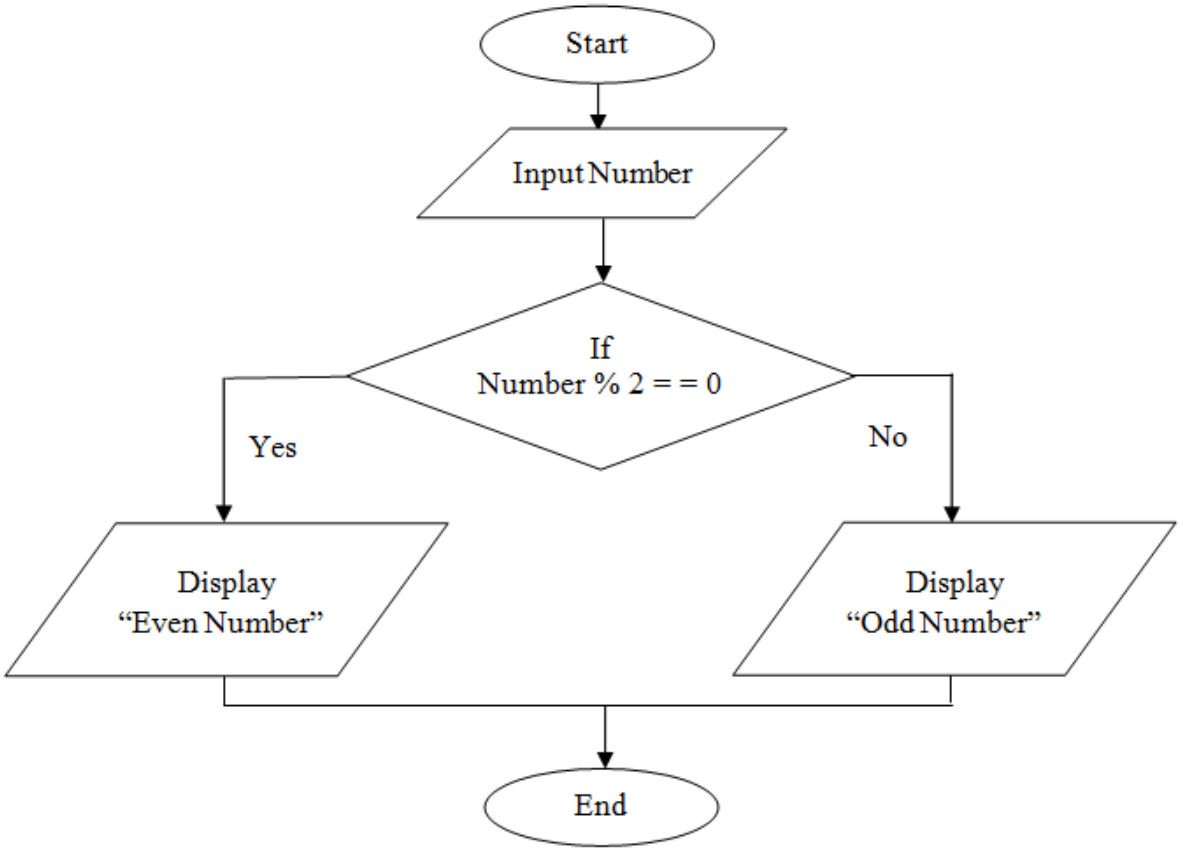
Ques4. Write a flowchart and pseudocode to convert temperature from Fahrenheit (℉) to Celsius (℃).



Pseudocode

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

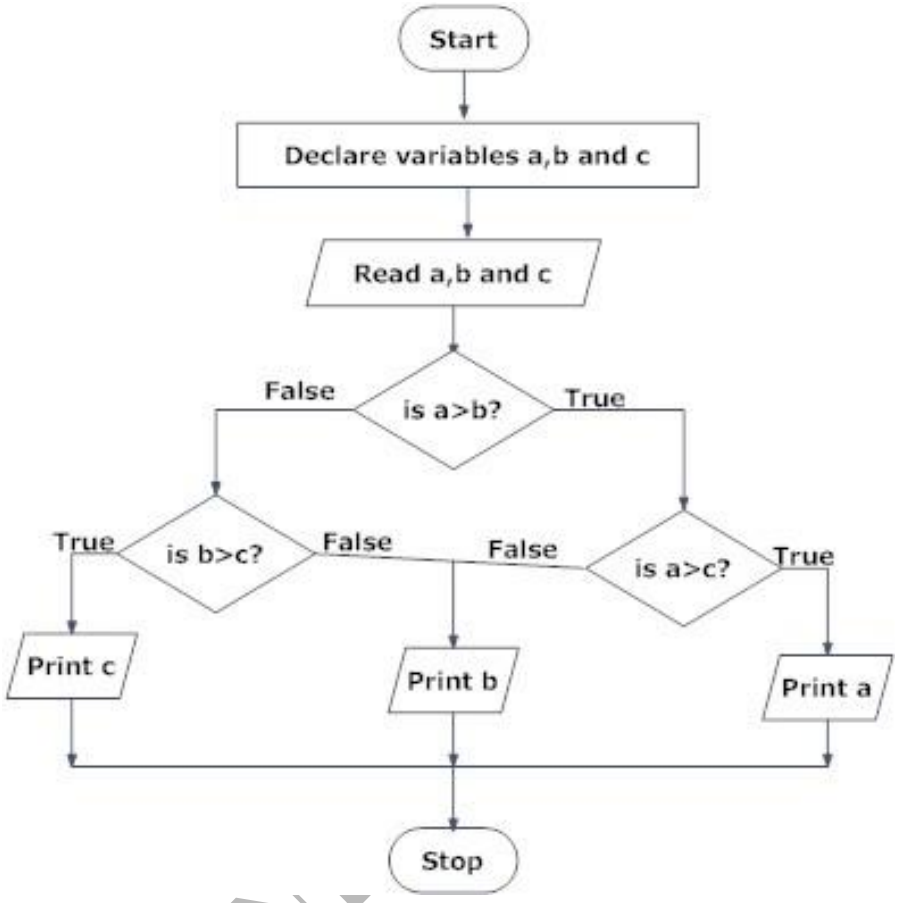
Ques5. Write a flowchart and pseudocode to check if a number is odd or even.



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

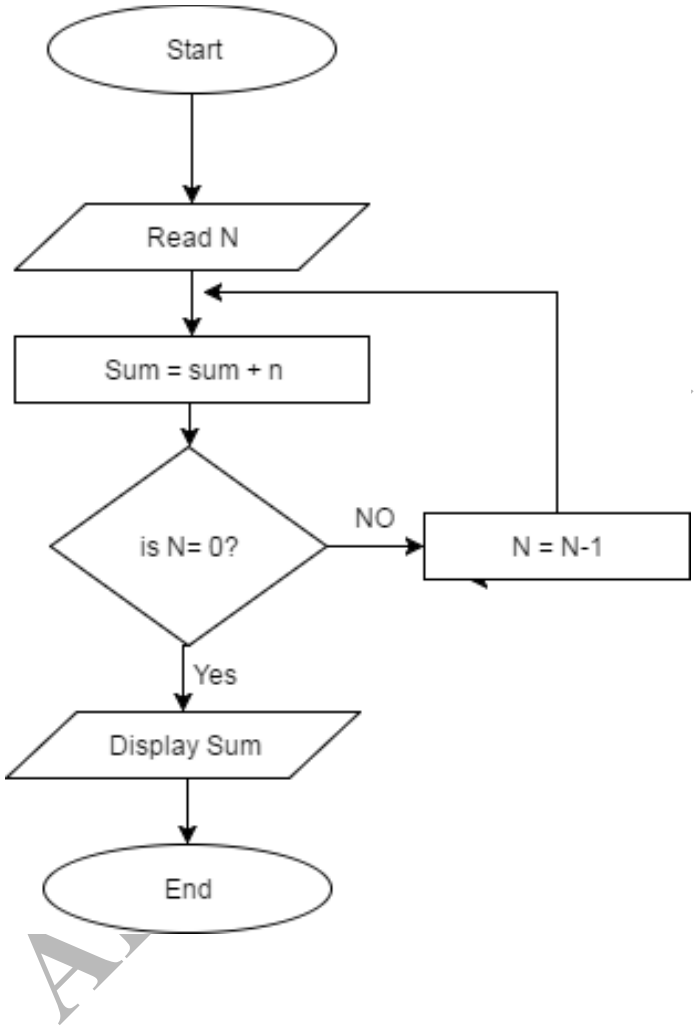
Ques6. Write a flowchart and pseudocode to find the max of 3 numbers.



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

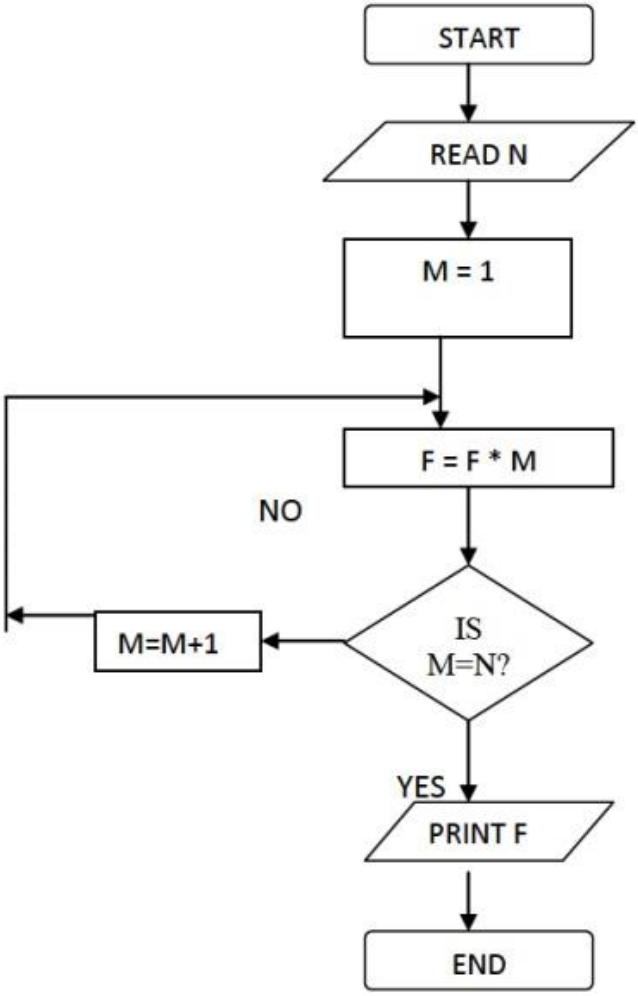
Ques7. Write a flowchart and pseudocode to find the sum of n natural numbers.



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

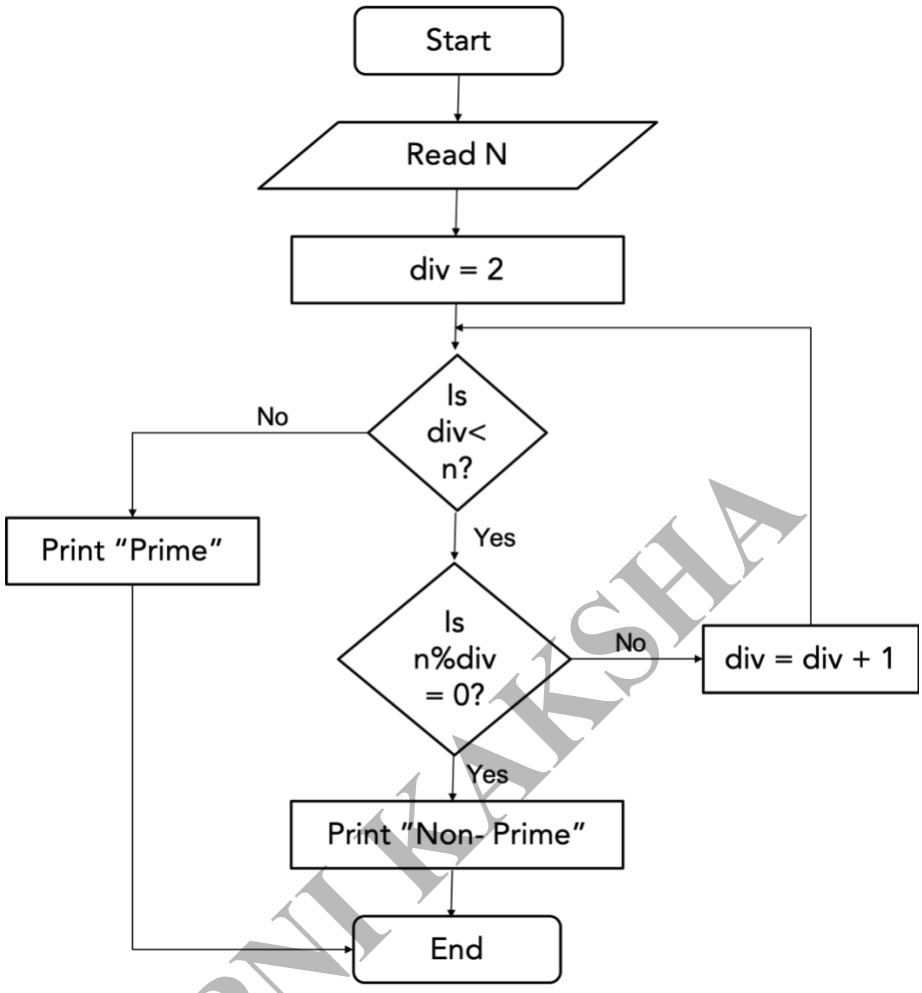
Ques8. Write a flowchart and pseudocode to find the factorial of a number.



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

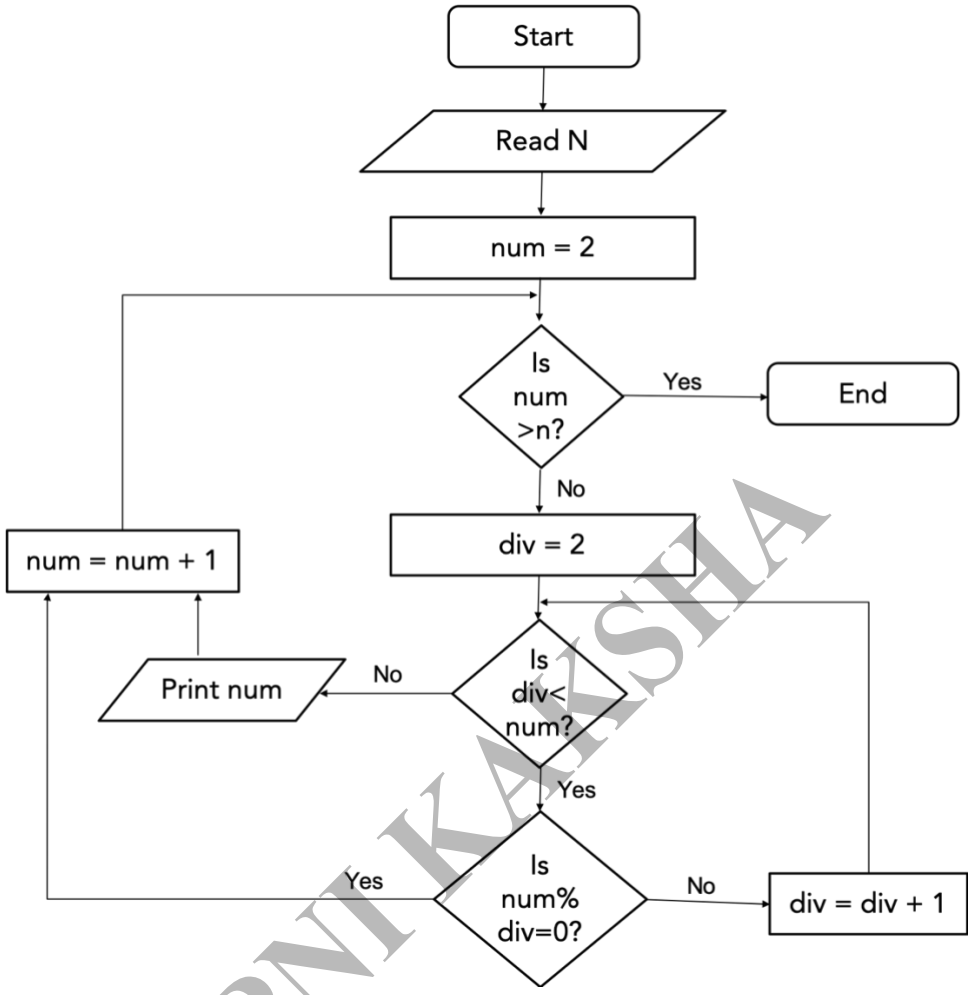
Ques9. Write a flowchart and pseudocode to find if a number is prime or not.



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.



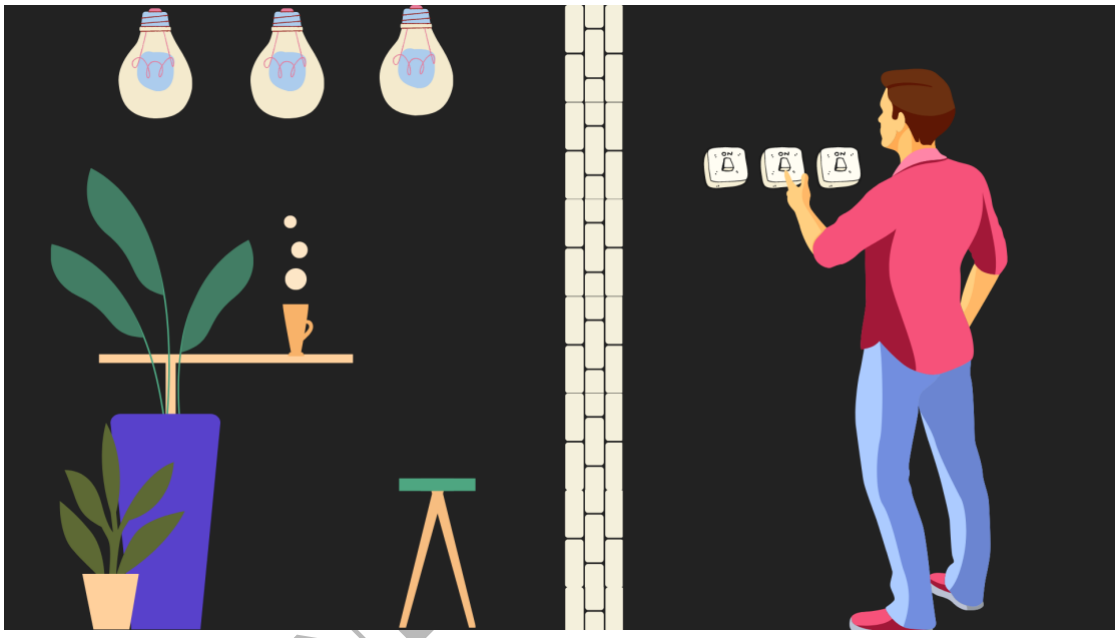
Pseudocode

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

Brain Teasers

1. You are standing outside of a room with no windows. The room has three light bulbs and three switches outside of the room. Each switch controls one of the light bulbs. You may only enter the room one time. How can you find out what switch goes to each light bulb?

(asked in LinkedIn)



Solution

To find out what switch goes to what light bulb, you could turn on the first switch and wait five minutes.

After five minutes have passed, turn off the first switch and turn on the second switch.

Look for the warm light bulb when you walk back into the room.

The light that is on goes to the second switch, the warm light bulb goes to the first switch and the last light bulb goes to the third switch.

2. You have a 3 gallon jug and 5 gallon jug, how do you measure out exactly 4 gallons?

(asked in Google)

Solution

First fill the 3 gallon jug. Then pour the 3 gallons into the 5 gallon jug.

Now the 3 gallon jug is empty, and the 5 gallon jug has 3 gallons in it.

Fill the 3 gallon jug again and pour into the 5 gallon jug. Only 2 gallons will fit because it already has 3. Exactly 1 gallon is left in the 3 gallon jug.

Dump out the 5 gallon jug.

Pour your 1 gallon into the 5 gallon jug. Fill up the 3 gallon jug one more time and pour it into the 5 gallon jug. You have exactly 4 gallons!

3. You're about to get on a plane to Seattle. You want to know if it's raining. You call 3 random friends who live there and ask each if it's raining. Each friend has a 2/3 chance of telling you the truth and a 1/3 chance of messing with you by lying. All 3 friends tell you that "Yes" it is raining. What is the probability that it's actually raining in Seattle?

(asked in Facebook)

Solution

You only need 1 of your friends to be telling the truth for it to be raining in Seattle.

It's fastest just to calculate the odds that all 3 are lying, and it's not raining.

Each friend has a 1/3 chance of lying. Multiply the odds together… you get 1/27 (1/3 * 1/3 * 1/3).

So 1/27 is the probability that all 3 friends lied at the same time.

The probability that at least 1 told you the truth? 26/27 or around a 96% that it's raining in Seattle.

Variables

A variable is a container (storage area) used to hold data.
Each variable should be given a unique name (identifier).

int a=2;

Here a is the variable name that holds the integer value 2.
The value of a can be changed, hence the name variable.

There are certain rules for naming a variable in C++

1. Can only have alphabets, numbers and underscore.
2. Cannot begin with a number.
3. Cannot begin with an uppercase character.
4. Cannot be a keyword defined in C++ language (like int is a keyword).

Fundamental Data Types in C++

Data types are declarations for variables. This determines the type and size of data associated with variables which is essential to know since different data types occupy different size of memory.

Data Type	Meaning	Size (in Bytes)
int	Integer	4
float	Floating-point	4
double	Double Floating-point	8
char	Character	1
wchar_t	Wide Character	2
bool	Boolean	1
void	Empty	0

1. int

- This data type is used to store integers.
- It occupies 4 bytes in memory.
- It can store values from -2147483648 to 2147483647.

Eg. int age = 18

2. float and double

- Used to store floating-point numbers (decimals and exponentials)
- Size of float is 4 bytes and size of double is 8 bytes.
- Float is used to store upto 7 decimal digits whereas double is used to store upto 15 decimal digits.

Eg. float pi = 3.14
double distance = 24E8 // 24 x 10^8

3. char

- This data type is used to store characters.
- It occupies 1 byte in memory.
- Characters in C++ are enclosed inside single quotes ‘ ‘.
- ASCII code is used to store characters in memory.

Eg. char ch = ‘a’;

ASCII Table

0	?	1	?	2	?	3	?	4	?	5	?	6	?	7	?
8	?	9	?	10	?	11	?	12	?	13	?	14	?	15	?
16	?	17	?	18	?	19	?	20	?	21	?	22	?	23	?
24	?	25	?	26	?	27	?	28	?	29	?	30	?	31	?
32		33	!	34	"	35	#	36	$	37	%	38	&	39	'
40	(	41	)	42	*	43	+	44	,	45	-	46	.	47	/
48	0	49	1	50	2	51	3	52	4	53	5	54	6	55	7
56	8	57	9	58	:	59	;	60	<	61	=	62	>	63	?
64	@	65	A	66	B	67	C	68	D	69	E	70	F	71	G
72	H	73	I	74	J	75	K	76	L	77	M	78	N	79	O
80	P	81	Q	82	R	83	S	84	T	85	U	86	V	87	W
88	X	89	Y	90	Z	91	[	92	\	93	]	94	^	95	_
96	`	97	a	98	b	99	c	100	d	101	e	102	f	103	g
104	h	105	i	106	j	107	k	108	l	109	m	110	n	111	o
112	p	113	q	114	r	115	s	116	t	117	u	118	v	119	w
120	x	121	y	122	z	123	{	124	|	125	}	126	~		

4. bool

- This data type has only 2 values – true and false.
- It occupies 1 byte in memory.
- True is represented as 1 and false as 0.

Eg. bool flag = false

C++ Type Modifiers

Type modifiers are used to modify the fundamental data types.

Data Type	Size (in Bytes)	Meaning
signed int	4	used for integers (equivalent to int)
unsigned int	4	can only store positive integers
short	2	used for small integers (range -32768 to 32767)
long	at least 4	used for large integers (equivalent to long int)
long long int	8	used for very large integers (equivalent to long long int).
unsigned long long	8	used for very large positive integers or 0 (equivalent to unsigned long long int)
long double	8	used for large floating-point numbers
signed char	1	used for characters (guaranteed range -127 to 127)
unsigned char	1	used for characters (range 0 to 255)

Derived Data Types

These are the data types that are derived from fundamental (or built-in) data types. For example, arrays, pointers, function, reference.

User-Defined Data Types

These are the data types that are defined by user itself.
For example, class, structure, union, enumeration, etc.

We will be studying the derived and user-defined data types in detail in the further video lectures.

Hello World Program

```
// Hello world program in C++

#include<iostream>
using namespace std;

int main()
{
    cout << "Hello World!\n";
    return 0;
}
```

1. Comments
   The two slash(//) signs are used to add comments in a program. It does not have any effect on the behaviour or outcome of the program. It is used to give description of the program you're writing.

2. #include<iostream>
   #include is the pre-processor directive that is used to include files in our program. Here we are including the iostream standard file which is necessary for the declarations of basic standard input/output library in C++.

3. Using namespace std
   All elements of the standard C++ library are declared within namespace. Here we are using std namespace.

4. int main()
   The execution of any C++ program starts with the main function, hence it is necessary to have a main function in your program. 'int' is the return value of this function. (We will be studying about functions in more detail later).

5. {}
   The curly brackets are used to indicate the starting and ending point of any function. Every opening bracket should have a corresponding closing bracket.

6. cout<<”Hello World!\n”;
   This is a C++ statement. **cout** represents the standard output stream in C++. It is declared in the iostream standard file within the std namespace. The text between quotations will be printed on the screen.
   \n will not be printed, it is used to add line break.
   Each statement in C++ ends with a semicolon (**;**)

7. return 0;
   return signifies the end of a function. Here the function is main, so when we hit return 0, it exits the program. We are returning 0 because we mentioned the return type of main function as integer (int main). A zero indicates that everything went fine and a one indicates that something has gone wrong.

Input and Output in C++

The header file iostream must be included to make use of the input/output (cin/cout) operators.

Standard Output (cout)

By default, the standard output of a program points at the screen. So with the cout operator and the “insertion” operator (<<) you can print a message onto the screen.

```
#include<iostream>
 using namespace std;

 int main()
 {
   cout << "Hello World!";
   return 0;
 }
```

To print the content of a variable the double quotes are not used.

```
#include<iostream>
```

```
using namespace std;

int main()
{
    char Yes = 'y';
    cout << Yes;
    return 0;
}
```

The << operator can be used multiple times in a single statement. Take a look at an example:

```
#include<iostream>
using namespace std;

int main()
{
    cout << "Hello, " << "this is a test " << "string.";
    return 0;
}
```

It is possible to combine variables and text:

```
#include<iostream>
using namespace std;

int main()
{
    char Yes = 'y';
    cout << "Print the character " << Yes;
    return 0;
}
```

The cout operator does not put a line break at the end of the output. So if you want to print two sentences you will have to use the new-line character (\n).

```
cout << "This is one sentence.\n";
cout << "This is another.\n";
```

It is possible to use the **endl** manipulator instead of the new-line character.

```
cout << "This is one sentence." << endl;
cout << "This is another." << endl;
```

Standard input (cin)

In most cases the standard input device is the keyboard. With the cin and >> operators it is possible to read input from the keyboard.

Take a look at an example:

```
#include<iostream>
using namespace std;

 int main()
 {
   char ch;
   cout << "Press a character and press return: ";
   cin >> ch;
   cout << ch;
   return 0;
```

```
}
```

Note: The input is processed by cin after the return key is pressed.

The cin operator will always return the variable type that you use with cin. So if you request an integer you will get an integer and so on. This can cause an error when the user of the program does not return the type that you are expecting. (Example: you ask for an integer and you get a string of characters.)

The cin operator is also chainable. For example:

```
cin >> X >> Y;
```

In this case the user must give two input values, that are separated by any valid blank separator (tab, space or new-line).

Decision making

if/else

The if block is used to specify the code to be executed if the condition specified in if is true, the else block is executed otherwise.

```
#include <iostream>
using namespace std ;

int main () {
      int age ;
      cin >> age ;

      if ( age >= 18 ) {
           cout << "You can vote." ;
      }
      else {
           cout << "Not eligible for voting." ;
      }

      return 0 ;
}
```

---

else if

To specify multiple if conditions, we first use if and then the consecutive statements use else if.

```
#include <iostream>
using namespace std ;

int main () {
      int x,y ;
      cin >> x >> y ;

      if ( x == y ) {
          cout << "Both the numbers are equal" ;
      }
      else if ( x > y ) {
          cout << "X is greater than Y" ;
      }
      else {
           cout << "Y is greater than X" ;
```

```
    }

    return 0 ;
}
```

---

nested if

To specify conditions within conditions we make the use of nested ifs.

```
#include <iostream>
using namespace std ;

int main () {
    int x,y ;
    cin >> x >> y ;

    if ( x == y ) {
        cout << "Both the numbers are equal" ;
    }
    else {
        if ( x > y ) {
            cout << "X is greater than Y" ;
        }
        else {
            cout << "Y is greater than X" ;
        }
    }

    return 0 ;
}
```

Problems

1. Program to check if a number is even or odd.

```
#include<iostream>
using namespace std;

int main(){

  int n;
  cin>>n;

  if(n%2==0){
     cout<<"Even"<<endl;
  }
  else{
     cout<<"Odd"<<endl;
  }

   return 0;
}
```

2. Program to find maximum, minimum among two numbers.

```
#include<iostream>
using namespace std;

int main(){

  int n1,n2;
```

```
  cin>>n1>>n2;

  int max, min;

  if(n1>n2){
     max=n1;
     min=n2;
  }
  else{
     max=n2;
     min=n1;
  }

  cout<<"Max= "<<max<<endl;
  cout<<"Min= "<<min<<endl;

   return 0;
}
```

3. Program to find the maximum among three numbers.

```
#include<iostream>
using namespace std;

int main(){

  int a,b,c;
  cin>>a>>b>>c;

  if(a>b){
```

```
    if(a>c){
        cout<<a<<endl;
    }
    else{
        cout<<c<<endl;
    }
  }
  else{
    if(b>c){
        cout<<b<<endl;
    }
    else{
        cout<<c<<endl;
    }
  }

   return 0;
}
```

4. Program to check if a triangle is scalene, isosceles or equilateral.

```
#include <iostream>
using namespace std;

int main()
{

   int sidea, sideb, sidec;
```

```
    cout << "Input three sides of triangle: \n ";
    cin >> sidea >> sideb >> sidec;

    if (sidea == sideb && sideb == sidec)
    {
        cout << "This is an equilateral triangle. \n ";
    }
    else if (sidea == sideb || sidea == sidec || sideb == sidec)
    {
        cout << "This is an isosceles triangle. \n ";
    }
    else
    {
        cout << "This is a scalene triangle. \n ";
    }

    return 0;
}
```

5. Program to check if an alphabet is a vowel or a consonant.

```
#include <iostream>
using namespace std;

int main()
{
    char c;
    int isLowercaseVowel, isUppercaseVowel;
```

```
    cout << "Enter an alphabet: ";
    cin >> c;

    isLowercaseVowel = (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u');

    isUppercaseVowel = (c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U');

    if (isLowercaseVowel || isUppercaseVowel)
        cout << c << " is a vowel.";
    else
        cout << c << " is a consonant.";

    return 0;
}
```

Loops in C++

A loop is used for executing a block of statements repeatedly until a particular condition is satisfied. A loop consists of an initialization statement, a test condition and an increment statement.

for loop

The syntax of the for loop is

```
for (initialization; condition; update) {
    // body of-loop
}
```

```
#include<iostream>
using namespace std;

int main(){

   for(int i=1;i<=5;i++){
       cout<<i<<" ";
   }

   return 0;
}
```

Output –

```
URVIs-MacBook-Air:helloworld urvigoel$ ./forloop
1 2 3 4 5
```

The for loop is initialized by the value 1, the test condition is i<=5 i.e the loop is executed till the value of i remains lesser than or equal to 5. In each iteration the value of i is incremented by one by doing i++.

while loop

The syntax for while loop is

```
while (condition) {
    // body of the loop
}
```

```
#include<iostream>
using namespace std;

int main(){

    int i=1;

    while(i<=5){
        cout<<i<<" ";
        i++;
    }
    return 0;
}
```

Output-

```
URVIs-MacBook-Air:helloworld urvigoel$ ./whileloop
1 2 3 4 5
```

The while loop is initialized by the value 1, the test condition is i<=5 i.e the loop is executed till the value of i remains lesser than or equal to 5. In each iteration the value of i is incremented by one by doing i++.

do....while loop

The syntax for while loop is

```
do {
  // body of loop;
}
while (condition);
```

```
#include<iostream>
using namespace std;

int main(){

   int i=1;

   do
   {
      cout<<i<<" ";
      i++;
   } while (i<=5);

   return 0;
}
```

Output-

```
URVIs-MacBook-Air:helloworld urvigoel$ ./dowhile
1 2 3 4 5
```

The do while loop variable is initialized by the value 1, in each iteration the value of i is incremented by one by doing i++, the test condition is i<=5 i.e the loop is executed till the value of i remains lesser than or equal to 5. Since the testing condition is checked only once the loop has already run so a do while loop runs at least once.

Examples –

Ques1. Program to find sum of natural numbers till n.

```
#include<iostream>
using namespace std;

int main(){

    int n;
    cin>>n;

    int sum=0;
    for(int counter=1;counter<=n;counter++){
        sum=sum+counter;
    }

    cout<<sum<<endl;

    return 0;
}
```

Ques2. Program to display multiplication table upto 10.

```
#include <iostream>
using namespace std;

int main()
{
    int n;

    cout << "Enter a positive integer: ";
    cin >> n;

    for (int i = 1; i <= 10; ++i) {
        cout << n << " * " << i << " = " << n * i << endl;
    }

    return 0;
}
```

Ques3. Program to add only positive numbers.

```
#include <iostream>
using namespace std;

int main() {
    int number;
    int sum = 0;

    cout << "Enter a number: ";
    cin >> number;
```

```
    while (number >= 0) {

        sum += number;
        cout << "Enter a number: ";
        cin >> number;
    }

    cout << "\nThe sum is " << sum << endl;

    return 0;
}
```