UNIT 1

Introduction:

Programmers write instructions in various programming languages to perform their computationtasks such as:

- Machine level Language
- Assembly level Language
- High level Language

➢ Machine level Language :

Machine code or machine language is a set of instructions executed directly by a computer's central processing unit (CPU). Each instruction performs a very specific task, such as a load, a jump, or an ALU operation on a unit of data in a CPU register or memory. Every program directly executed by a CPU is made up of a series of such instructions.

➢ Assembly level Language :

An assembly language (or assembler language) is a low-level programming language for a computer, or other programmable device, in which there is a very strong (generally one-to-one) correspondence between the language and the architecture's machine code instructions. Assembly language is converted into executable machine code by a utility program referred to as an assembler; the conversion process is referred to as assembly, or assembling the code.

➢ High level Language :

High-level language is any programming language that enables development of a program in much simpler programming context and is generally independent of the computer's hardware architecture. High-level language has a higher level of abstraction from the computer, and focuses more on the programming logic rather than the underlying hardware components such as memory addressing and register utilization.

The first high-level programming languages were designed in the 1950s. Now there are dozens of different languages, including Ada , Algol, BASIC, COBOL, C, C++, JAVA, FORTRAN, LISP, Pascal, and Prolog. Such languages are considered high-level because they are closer to human languages and farther from machine languages. In contrast, assembly languages are considered low- level because they are very close to machine languages.

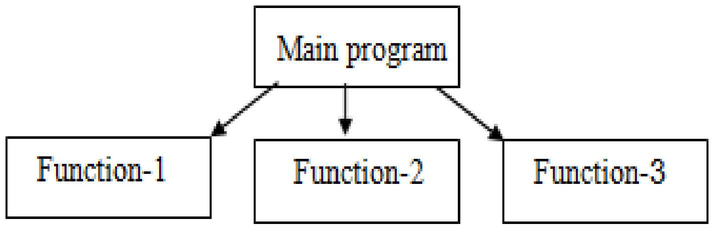
The high-level programming languages are broadly categorized in to two categories:

- Procedure oriented programming (POP) language.
- Object oriented programming (OOP) language.

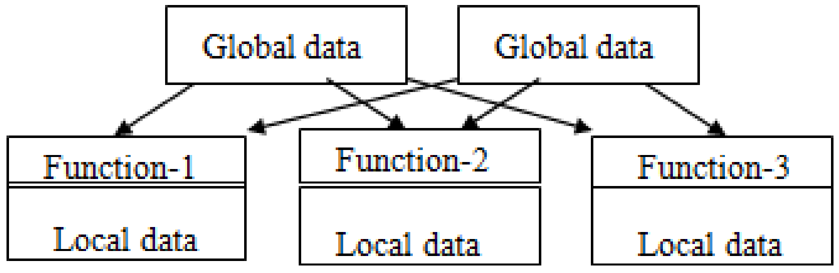
- **Procedure Oriented Programming Language (POP)**

In the procedure oriented approach, the problem is viewed as sequence of things to be done such as reading, calculation and printing.

Procedure oriented programming basically consist of writing a list of instruction or actions for the computer to follow and organizing these instruction into groups known as functions.



The disadvantage of the procedure oriented programming languages is:
1. Global data access
2. It does not model real word problem very well
3. No data hiding

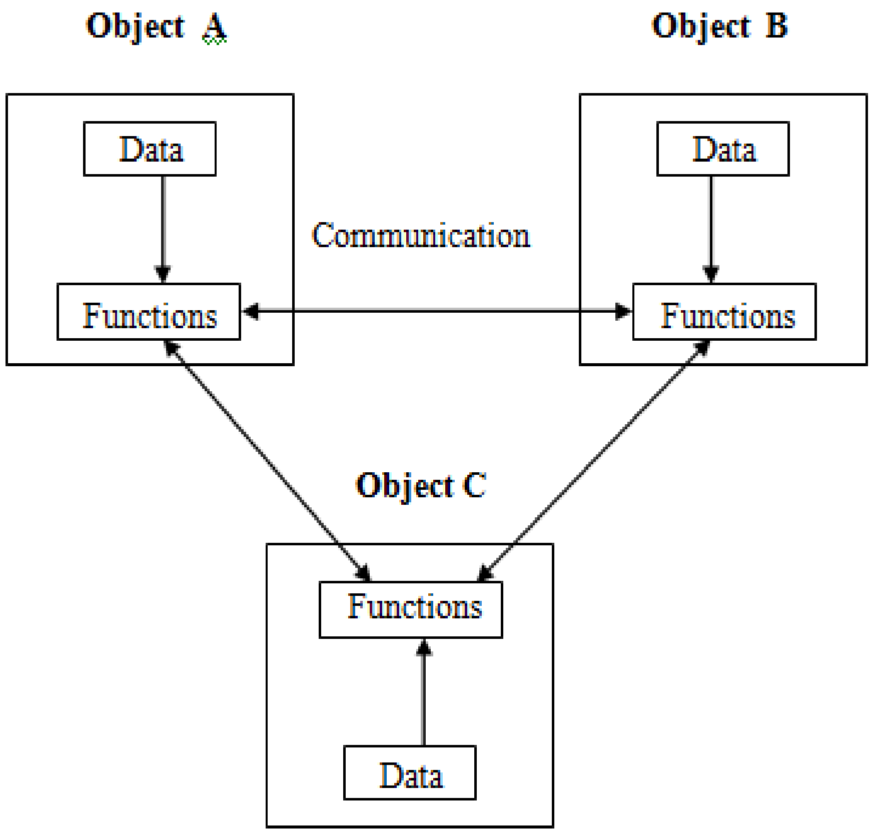


Characteristics of procedure oriented programming:

1. Emphasis is on doing things (algorithm)
2. Large programs are divided into smaller programs known as functions.
3. Most of the functions share global data
4. Data move openly around the system from function to function
5. Function transforms data from one form to another.
6. Employs top-down approach in program design.

- **Object Oriented Programming Language (OOP)**

"Object oriented programming as an approach that provides a way of modularizing programs by creating partitioned memory area for both data and functions that can be used as templates for creating copies of such modules on demand".
Writing object-oriented programs involves creating classes, creating objects from those classes, and creating applications, which are stand-alone executable programs that use those objects. After being created, classes can be reused over and over again to develop new programs. Thinking in an object-oriented manner involves envisioning program components as objects that belong to classes and are similar to concrete objects in the real world; then, you can manipulate the objects and have them interrelate with each other to achieve a desired result.



➢ **Features of the Object Oriented programming-**

1. Emphasis is on doing rather than procedure.
2. Programs are divided into what are known as objects.
3. Data structures are designed such that they characterize the objects.
4. Functions that operate on the data of an object are tied together in the data structure.
5. Data is hidden and can't be accessed by external functions.
6. Objects may communicate with each other through functions.
7. New data and functions can be easily added.
8. Follows bottom-up approach in program design.

➢ Need for Object oriented Programming:

Object-oriented programming scales very well, from the most trivial of problems to the most complex tasks. It provides a form of abstraction that resonates with techniques people use to solve problems in their everyday life.
Object-oriented programming was developed because limitations were discovered in earlier approaches to programming. There were two related problems. First, functions have unrestricted access to global data. Second, unrelated functions and data, the basis of the procedural paradigm provide a poor model of the real world.

➢ Benefits of Object oriented Programming:

1. Simplicity: Software objects model real world objects, so the complexity is reduced and the program structure is very clear.

2. Modularity: Each object forms a separate entity whose internal workings are decoupled from other parts of the system.

3. Modifiability: It is easy to make minor changes in the data representation or the procedures in an object-oriented program. Changes inside a class do not affect any other part of a program, since the only public interface that the external world has to a class is through the use of methods.

4. Extensibility: adding new features or responding to changing operating environments can be solved by introducing a few new objects and modifying some existing ones.

5. Maintainability: objects can be maintained separately, making locating and fixing problems easier.
6. Re-usability: objects can be reused in different programs.

➢ Application of OOPs :

The most popular application of oops up to now, has been in the area of user interface design such as windows. There are hundreds of windowing systems developed using OOPs techniques.

Real business systems are often much more complex and contain many more objects with complicated attributes and methods. OOPs are useful in this type of applications because it can simplify a complex problem. The promising areas for application of OOPs are:

1. Real – Time systems.
2. Simulation and modeling
3. Object oriented databases.
4. Hypertext, hypermedia and expertext.
5. Al and expert systems.
6. Neural networks and parallel programming.
7. Decision support and office automation systems.

- **Basics of C++ :**

C++ is an object oriented programming language. It was developed by Bjarne Stroustrup in 1979 at Bell Laboratories in Murray Hill, New Jersey. He initially called the new language "C with Classes." However, in 1983 the name was changed to C++.

C++ is a superset of C. Stroustrup built C++ on the foundation of C, including all of C’s features, attributes, and benefits. Most of the features that Stroustrup added to C were designed to support object-oriented programming .These features comprise of classes, inheritance, function overloading and operator overloading. C++ has many other new features as well, including an improved approach to input/output (I/O) and a new way to write comments.

C++ is used for developing applications such as editors, databases, personal file systems, networking utilities, and communication programs. Because C++ shares C’s efficiency, much high-performance systems software is constructed using C++.

➢ **C++ Comments:**

C++ introduces a new comment symbol //(double slash). Comments start with a double slash symbol and terminate at the end of line. A comment may start anywhere in the line and whatever follows till the end of line is ignored. Note that there is no closing symbol.

The double slash comment is basically a single line comment. Multi line comments can be written as follows:

```
// this is an example of

// c++ program

// thank you
```

The c comment symbols /* ….*/ are still valid and more suitable for multi line comments.

```
/* this is an example of c++ program */
```

Output Operator:

The statement cout <<”Hello, world” displayed the string with in quotes on the screen. The identifier cout can be used to display individual characters, strings and even numbers. It is a predefined object that corresponds to the standard output stream. Stream just refers to a flow of data and the standard Output stream normally flows to the screen display. The cout object, whose properties are defined in iostream.h represents that stream. The insertion operator << also called the ‘put to’ operator directs the information on its right to the object on its left.

Return Statement:

In C++, main ( ) returns an integer type value to the operating system. Therefore every main () in C++ should end with a return (0) statement, otherwise a warning or an error might occur.

Input Operator:

The statement cin>> number 1;

is an input statement and causes. The program to wait for the user to type in a number. The number keyed in is placed in the variable number1. The identifier cin is a predefined object in C++ that corresponds to the standard input stream. Here this stream represents the key board.

The operator >> is known as get from operator. It extracts value from the keyboard and assigns it to the variable on its right.

Cascading Of I/O Operator:

```
cout<<”sum=”<<sum<<”\n”;
cout<<”sum=”<<sum<<”\n”<<”average=”<<average<<”\n”;
cin>>number1>>number2;
```

- **Structure Of A Program :**

Probably the best way to start learning a programming language is by writing a program. Therefore, here is our first program:

```
// my first program in C++

#include <iostream> using namespace std;

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

Output:-Hello World!

The first panel shows the source code for our first program. The second one shows the result of the program once compiled and executed. The way to edit and compile a program depends on the compiler you are using. Depending on whether it has a Development Interface or not and on its version. Consult the compilers section and the manual or help included with your compiler if you have doubts on how to compile a C++ console program.

The previous program is the typical program that programmer apprentices write for the first time, and its result is the printing on screen of the "Hello World!" sentence. It is one of the simplest programs that can be written in C++, but it already contains the fundamental components that every C++ program has. We are going to look line by line at the code we have just written:

// my first program in C++

This is a comment line. All lines beginning with two slash signs (//) are considered comments and do not have any effect on the behavior of the program. The programmer can use them to include short explanations or observations within the source code itself. In this case, the line is a brief description of what our program is.

#include <iostream>
Lines beginning with a hash sign (#) are directives for the preprocessor. They are not regular code lines with expressions but indications for the compiler's preprocessor. In this case the directive #include<iostream> tells the preprocessor to include the iostream standard file. This specific file (iostream) includes the declarations of the basic standard input-output library in C++, and it is included because its functionality is going to be used later in the program.

using namespace std;
All the elements of the standard C++ library are declared within what is called a namespace, the namespace with the name std. So in order to access its functionality we declare with this expression that we will be using these entities. This line is very frequent in C++ programs that use the standard library, and in fact it will be included in most of the source codes included in these tutorials.

int main ()
This line corresponds to the beginning of the definition of the main function. The main function is the point by where all C++ programs start their execution, independently of its location within the source code. It does not matter whether there are other functions with other names defined before or after it – the instructions contained within this function's definition will always be the first ones to be executed in any C++ program. For that same reason, it is essential that all C++ programs have a main function.
The word main is followed in the code by a pair of parentheses (()). That is because it is a function declaration: In C++, what differentiates a function declaration from other types of expressions are these parentheses that follow its name. Optionally, these parentheses may enclose a list of parameters within them.
Right after these parentheses we can find the body of the main function enclosed in braces ({}). What is contained within these braces is what the function does when it is executed.

cout << "Hello World!";
This line is a C++ statement. A statement is a simple or compound expression that can actually produce some effect. In fact, this statement performs the only action that generates a visible effect in our first program.
cout represents the standard output stream in C++, and the meaning of the entire statement is to insert a sequence of characters (in this case the Hello World sequence of characters) into the standard output stream (which usually is the screen).
cout is declared in the iostream standard file within the std namespace, so that's why we needed to include that specific file and to declare that we were going to use this specific namespace earlier in our code.
Notice that the statement ends with a semicolon character (;). This character is used to mark the end of the statement and in fact it must be included at the end of all expression statements in all C++ programs (one of the most common syntax errors is indeed to forget to include some semicolon after a statement).

return 0;
The return statement causes the main function to finish. return may be followed by a return code (in our example is followed by the return code 0). A return code of 0 for the main function is generally interpreted as the program worked as expected without any errors during its execution. This is the most usual way to end a C++ console program.
You may have noticed that not all the lines of this program perform actions when the code is executed. There were lines containing only comments (those beginning by //). There were lines with directives for the compiler's preprocessor (those beginning by #). Then there were lines that began the declaration of a function (in this case, the main function) and, finally lines with statements (like the insertion into cout), which were all included within the block delimited by the braces ({}) of the main function.
The program has been structured in different lines in order to be more readable, but in C++, we do not have strict rules on how to separate instructions in different lines. For example, instead of

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

We could have written:

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

All in just one line and this would have had exactly the same meaning as the previous code.
In C++, the separation between statements is specified with an ending semicolon (;) at the end of each one, so the separation in different code lines does not matter at all for this purpose. We can write many statements per line or write a single statement that takes many code lines. The division of code in different lines serves only to make it more legible and schematic for the humans that may read it.

Let us add an additional instruction to our first program:

```
// my second program in C++ #include <iostream>
using namespace std;

int main ()
{
cout << "Hello World! ";
cout << "I'm a C++ program";
return 0;
}
```

```
Output:-Hello World! I'm a C++ program
```

In this case, we performed two insertions into cout in two different statements. Once again, the separation in different lines of code has been done just to give greater readability to the program, since main could have been perfectly valid defined this way:

```
int main ()
{
 cout << " Hello World! ";
 cout << " I'm a C++ program ";
 return 0;
}
```

We were also free to divide the code into more lines if we considered it more convenient: int main ()

```
{
 cout << "Hello World!";
 cout << "I'm a C++ program";
 return 0;
}
```

And the result would again have been exactly the same as in the previous examples.
Preprocessor directives (those that begin by #) are out of this general rule since they are not statements. They are lines read and processed by the preprocessor and do not produce any code by themselves. Preprocessor directives must be specified in their own line and do not have to end with a semicolon (;).

- **The general structure of C++ program with classes is shown as:**

1. Documentation Section
2. Preprocessor Directives or Compiler Directives Section
   (i) Link Section
   (ii) Definition Section
3. Global Declaration Section
4. Class declaration or definition
5. Main C++ program function called main ( )

- **Tokens:**

The smallest individual units in program are known as tokens. C++ has the following tokens.

i. Keywords
ii. Identifiers
iii. Constants
iv. Strings
v. Operators

- **Keywords:**

The keywords implement specific C++ language feature. They are explicitly reserved identifiers and can't be used as names for the program variables or other user defined program elements. The keywords not found in ANSI C are shown in red letter.

➢ **C++ keywords:**

When a language is defined, one has to design a set of instructions to be used for communicating with the computer to carry out specific operations. The set of instructions which are used in programming, are called keywords. These are also known as reserved words of the language. They have a specific meaning for the C++ compiler and should be used for giving specific instructions to the computer. These words cannot be used for any other purpose, such as naming a variable. C++ is a case-sensitive language, and it requires that all keywords be in lowercase. C++ keywords are:

Asm	Double	new	switch
Auto	Else	operator	template
Break	Enum	private	this
Case	Extern	protected	throw
Catch	Float	public	try
Char	For	register	typedef
Class	Friend	return	union
Const	Goto	short	unsigned
Continue	If	signed	virtual
Default	Inline	sizeof	void

Delete	Long	struet	while

- **Identifiers:**

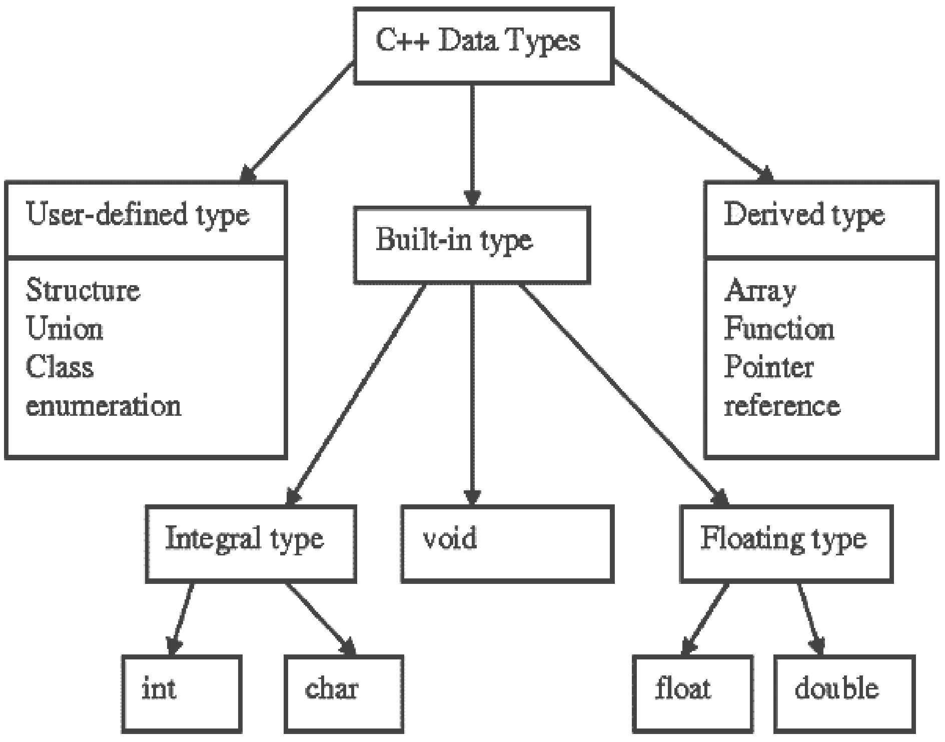
Identifiers refers to the name of variable , functions, array, class etc. created by programmer. Each language has its own rule for naming the identifiers.

The following rules are common for both C and C++.

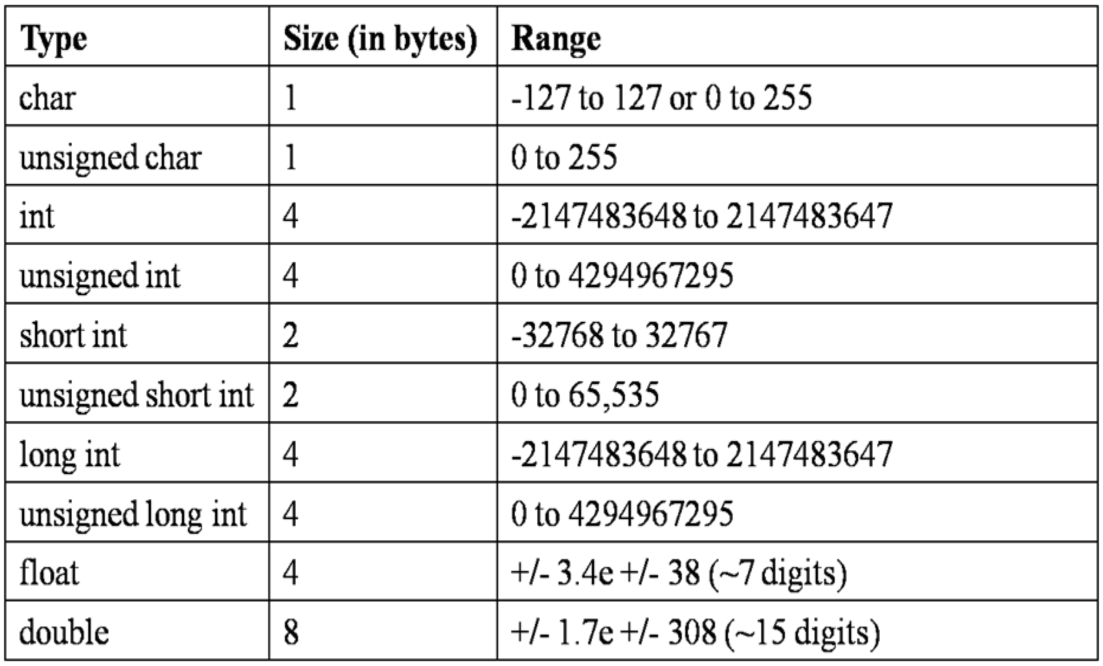
1. Only alphabetic chars, digits and underscore are permitted.
2. The name can't start with a digit.
3. Upper case and lower case letters are distinct.
4. Keywords can't be used as Identifiers.
5. No space is allowed between names.

- **Data types :**

Data type defines size and type of values that a variable can store along with the set of operations that can be performed on that variable. C++ provides built-in data types that correspond to integers, characters, floating-point values, and Boolean values. There are the seven basic data types in C++ as shown below:



C++ allows certain of the basic types to have modifiers preceding them. A modifier alters the meaning of the base type so that it more precisely fits the needs of various situations. The data type modifiers are: signed, unsigned, long and short.



Type	Size (in bytes)	Range
char	1	-127 to 127 or 0 to 255
unsigned char	1	0 to 255
int	4	-2147483648 to 2147483647
unsigned int	4	0 to 4294967295
short int	2	-32768 to 32767
unsigned short int	2	0 to 65,535
long int	4	-2147483648 to 2147483647
unsigned long int	4	0 to 4294967295
float	4	+/- 3.4e +/- 38 (~7 digits)
double	8	+/- 1.7e +/- 308 (~15 digits)

- **User defined data types:**

  - **Structers and classes**

We have used user defined data types such as struct, and union in C. While these more features have been added to make them suitable for object oriented programming. C++ also permits us to define another user defined data type known as class which can be used just like any other basic data type to declare a variable. The class variables are known as objects, which are the central focus of oops.

  - **Enumerated data type:**

An enumerated data type is another user defined type which provides a way for attaching names to number, these by increasing comprehensibility of the code. The enum keyword automatically enumerates a list of words by assigning them values 0,1,2 and soon. This facility provides an alternative means for creating symbolic.

Example:

enum shape {

circle,square,triangle}

enum

colour{red,blue,green,yell

ow}enum position {off,on}

The enumerated data types differ slightly in C++ when compared with ANSI C. In C++, the tag names shape, colour, and position become new type names. That means we can declare new variables using the tag names.

Example:
Shape ellipse; //ellipse is of type shape

colour background ; // back ground is of type colour

ANSI C defines the types of enums to be ints. In C++,each enumerated data type retains its own separate type. This means that C++ does not allow an int value to be automatically converted toan enum.

Example:
colour background =blue; //vaid

colour background =7; //error in c++

colour background =(colour) 7;//ok

How ever an enumerated value can be used in place of an int value.

Example:
int c=red ;//valid, colour type promoted to int

By default, the enumerators are assigned integer values starting with 0 for the first enumerator, 1 for the second and so on. We can also write

enum color {red, blue=4,green=8};

enum color {red=5,blue,green};

C++ also permits the creation of anonymous enums (i.e, enums without tag names)

Example:
enum{off,on};

Here off is 0 and on is 1.these constants may be referenced in the same manner as regular constants.

Example:
int switch-1=off;

int switch-2=on;

- **Constants :**

Constants refer to fixed values that the program cannot alter. Constants can be of any of the basic data types. The way each constant is represented depends upon its type. Constants are also called literals. We can use keyword const prefix to declare constants with a specific type as follows:
const type variableName = value;
e.g,

```
const int LENGTH = 10;
```

- **Declaration of variables:**

In C, all the variable which is to be used in programs must be declared at the beginning of the program. But in C++ we can declare the variables any whose in the program where it requires .This makes theprogram much easier to write and reduces the errors that may be caused by having to scan back and forth. It also makes the program easier to understand because the variables are declared in the context of their use.

Example:

```
main()
   {
   float x, average;
 float  sum=0;
    for (int i =1; i < 5; i++)
   {
    cin >> x;
    sum = sum+x
   }
    float average;
    average = sum/x;
    cout << average;
   }
```

- **Reference variables:**

C++interfaces a new kind of variable known as the reference variable. A references variable provides an alias.(alternative name) for a previously defined variable. For example ,if we make the variable sum a reference to the variable total, then sum and total can be used interchangeably to represent the variuble.

A reference variable is created as follows:

Syntax: Datatype & reference_name = variable name;

Example:

```
float total=1500;
float &sum=total;
```

Here sum is the alternative name for variables total, both the variables refer to the same data object in the memory .
A reference variable must be initialized at the time of declaration .
Note that C++ assigns additional meaning to the symbol & here & is not an address operator.
The notation float & means reference to float.

Example:

```
int n[10];
int &x=n[10];
char &a='\n';
```

- **Operators in c++ :**

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. C++ is rich in built-in operators. Generally, there are six type of operators: Arithmetical operators, Relational operators, Logical operators, Assignment operators, Conditional operators, Comma operator.

C++ has a rich set of operators. All C operators are valid in C++ also. In addition. C++ introduces some new operators.

<<	insertion operator
>>	extraction operator
: :	scope resolution operator
: :*	pointer to member declarator
*	pointer to member operator
.*	pointer to member operator
Delete	memory release operator
Endl	line feed operator
New	memory allocation operator
Setw	field width operator

- **Arithmetical operators :**

Arithmetical operators +, -, *, /, and % are used to performs an arithmetic (numeric) operation.

Operator	**Meaning**
+	Addition
-	Subtraction
*****	Multiplication
/	Division
%	Modulus

You can use the operators +, -, *, and / with both integral and floating-point data types. Modulus or remainder % operator is used only with the integral data type.

- **Relational operators :**

The relational operators are used to test the relation between two values. All relational operators are binary operators and therefore require two operands. A relational expression returns zero when the relation is false and a non-zero when it is true. The following table shows the relational operators.

Relational Operators	**Meaning**
<	Less than
<=	Less than or equal to
= =	Equal to
>	Greater than
>=	Greater than or equal to
! =	Not equal to

- **Logical operators :**

The logical operators are used to combine one or more relational expression. The logical operators are

Operators	**Meaning**
||	OR
&&	AND
!	NOT

- **Assignment operator :**

The assignment operator '=' is used for assigning a variable to a value. This operator takes the expression on its right-hand-side and places it into the variable on its left-hand-side. For example:
m = 5;

The operator takes the expression on the right, 5, and stores it in the variable on the left, m.
x = y = z = 32;

This code stores the value 32 in each of the three variables x, y, and z. In addition to standard assignment operator shown above, C++ also support compound assignment operators.

- **Compound Assignment Operators :**

Operator	**Example**	**Equivalent to**
+ =	A + = 2	A = A + 2
- =	A - = 2	A = A – 2
% =	A % = 2	A = A % 2
/=	A/ = 2	A = A / 2
*=	A * = 2	A = A * 2

- **Increment and Decrement Operators**

C++ provides two special operators viz '++' and '--' for incrementing and decrementing the value of a variable by 1. The increment/decrement operator can be used with any type of variable but it cannot be used with any constant. Increment and decrement operators each have two forms, pre and post.

The syntax of the increment operator is:
Pre-increment: ++variable
Post-increment: variable++

The syntax of the decrement operator is:
Pre-decrement: —variable
Post-decrement: variable—

- In Prefix form first variable is first incremented/decremented, then evaluated
- In Postfix form first variable is first evaluated, then incremented / decremented.

- **Conditional operator :**

The conditional operator ?: is called ternary operator as it requires three operands. The format of the conditional operator is :

Conditional_ expression ? expression1 : expression2;
If the value of conditional expression is true then the expression1 is evaluated, otherwise expression2 is evaluated.

int a = 5, b = 6;
big = (a > b) ? a : b;

The condition evaluates to false, therefore big gets the value from b and it becomes 6.

- **The comma operator :**

The comma operator gives left to right evaluation of expressions. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

int a = 1, b = 2, c = 3, i; // comma acts as separator, not as an operator

i = (a, b); // stores b into i would first assign the value of a to i, and then assign value of b to variable i. So, at the end, variable i would contain the value 2.

- **The sizeof operator :**

The sizeof operator can be used to find how many bytes are required for an object to store in memory. For example

sizeof (char) returns 1
sizeof (float) returns 4

- **Typecasting :**

Typecasting is the concept of converting the value of one type into another type. For example, you might have a float that you need to use in a function that requires an integer.

- **Implicit conversion :**

Almost every compiler makes use of what is called automatic typecasting. It automatically converts one type into another type. If the compiler converts a type it will normally give a warning. For example this warning: conversion from 'double' to 'int', possible loss of data. The problem with this is, that you get a warning (normally you want to compile without warnings and errors) and you are not in control. With control we mean, you did not decide to convert to another type, the compiler did. Also the possible loss of data could be unwanted.

- **Explicit conversion**

The C++ language have ways to give you back control. This can be done with what is called an explicit conversion.

  - **Four typecast operators**

  The C++ language has four typecast operators:

  - static_cast
  - reinterpret_cast
  - const_cast
  - dynamic_cast

- **Type Conversion :**

The Type Conversion is that which automatically converts the one data type into another but remember we can store a large data type into the other. For example we can't store a float into int because a float is greater than int.
When a user can convert the one data type into then it is called as the **type casting.** The type Conversion is performed by the compiler but a casting is done by the user for example converting a float into int. When we use the Type Conversion then it is called the promotion. In the type casting when we convert a large data type into another then it is called as the demotion. When we use the type casting then we can loss some data.

- **Control Statements :**

Control structures allow controlling the flow of program's execution based on certain conditions. C++ supports following basic control structures:

1. Selection Control statements
2. Iteration (Loop) Control Statements
3. Jump Statements

1) Selection Control statements :

Selection Control structures allows to control the flow of program's execution depending upon the state of a particular condition being true or false .C++ supports two types of selection statements :if and switch. Condition operator (?:) can also be used as an alternative to the if statement. There are various types of if statements in C++.

- if statement
- if-else statement
- nested if statement
- if-else-if ladder

➢ **IF Statement :**

The C++ if statement tests the condition. It is executed if condition is true.

```
if(condition){
//code to be executed
}
```

Example :

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

int main () {
  int num = 10;
        if (num % 2 == 0)
        {
           cout<<"It is even number";
        }
  return 0;
}
```

Output:

```
It is even number
```

- **IF-else Statement :**

The C++ if-else statement also tests the condition. It executes if block if condition is true otherwise else block is executed.

```
if(condition){
//code if condition is true
}else{
//code if condition is false
}
```

Example

```
#include <iostream>
using namespace std;
int main () {
  int num = 11;
        if (num % 2 == 0)
        {
           cout<<"It is even number";
        }
        else
        {
           cout<<"It is odd number";
        }
  return 0;
}
```

Output:

```
It is odd number
```

- **IF-else-if ladder Statement**

The C++ if-else-if ladder statement executes one condition from multiple statements.

```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Example

```
        #include <iostream>
        using namespace std;
        int main () {
             int num;
             cout<<"Enter a number to check grade:";
             cin>>num;
                if (num <0 || num >100)
                {
                   cout<<"wrong number";
                }
                else if(num >= 0 && num < 50){
                   cout<<"Fail";
                }
                else if (num >= 50 && num < 60)
                {
                   cout<<"D Grade";
                }
                else if (num >= 60 && num < 70)
                {
                   cout<<"C Grade";
                }
                else if (num >= 70 && num < 80)
                {
                   cout<<"B Grade";
                }
                else if (num >= 80 && num < 90)
                {
                   cout<<"A Grade";
                }
                else if (num >= 90 && num <= 100)
                {
                   cout<<"A+ Grade";
                }
           }
```

Output:

```
Enter a number to check grade:66
C Grade
```

Output:

```
Enter a number to check grade:-2
wrong number
```

- **Switch Statement :**

The C++ switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement in C++.

```
switch(expression){
case value1:
 //code to be executed;
 break;
case value2:
 //code to be executed;
 break;
......

default:
 //code to be executed if all cases are not matched;
 break;
}
```

Example

```
#include <iostream>
using namespace std;
int main () {
       int num;
       cout<<"Enter a number to check grade:";
       cin>>num;
           switch (num)
          {
              case 10: cout<<"It is 10"; break;
              case 20: cout<<"It is 20"; break;
              case 30: cout<<"It is 30"; break;
              default: cout<<"Not 10, 20 or 30"; break;
          }
    }
```

Output:

```
Enter a number:
10
It is 10
```

Output:

```
Enter a number:
55
Not 10, 20 or 30
```

2) Loop control Statements :

A loop statement allows us to execute a statement or group of statements multiple times. Loops or iterative statements tell the program to repeat a fragment of code several times or as long as a certain condition holds. C++ provides three convenient iterative statements: while, for, and do-while.

- **While loop :**

In C++, while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop than for loop.

```
while(condition){
//code to be executed
}
```

Example
Let's see a simple example of while loop to print table of 1.

```
#include <iostream>
using namespace std;
int main() {
 int i=1;
        while(i<=10)
      {
           cout<<i <<"\n";
           i++;
          }
    }
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

- **Do-While Loop :**

The C++ do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

The C++ do-while loop is executed at least once because condition is checked after loop body.

```
do{
//code to be executed
}while(condition);
```

Example

```
include <iostream>
using namespace std;
int main() {
    int i = 1;
        do{
            cout<<i<<"\n";
            i++;
        } while (i <= 10) ;
}
```

Output:

```
1
2
3
4
5
6
7
8
9
10
```

- **For Loop :**

The C++ for loop is used to iterate a part of the program several times. If the number of iteration is fixed, it is recommended to use for loop than while or do-while loops.

The C++ for loop is same as C/C#. We can initialize variable, check condition and increment/decrement value.

```
For (initialization; condition; incr/decr){
//code to be executed
}
```

Example

```
#include <iostream>
using namespace std;
int main() {
        for(int i=1;i<=10;i++){
           cout<<i <<"\n";
```

```
            }
        }
```
Output:

```
1
2
3
4
5
6
7
8
9
10
```

3) Jump Statements :

In C++, jump statements allows program to execute in a non-linear fashion.

- **Break Statement :**

The C++ break is used to break loop or switch statement. It breaks the current flow of the program at the given condition. In case of inner loop, it breaks only inner loop.

jump-statement;
break;

Example
Let's see a simple example of C++ break statement which is used inside the loop.

```
#include <iostream>
using namespace std;
int main() {
   for (int i = 1; i <= 10; i++)
       {
          if (i == 5)
          {
             break;
          }
     cout<<i<<"\n";
       }
}
```
Output:

```
1
2
3
4
```

- **Continue Statement :**

The C++ continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

```
jump-statement;
continue;
```

Example

```
#include <iostream>
using namespace std;
int main()
{
    for(int i=1;i<=10;i++){
        if(i==5){
            continue;
        }
        cout<<i<<"\n";
      }
}
```

Output:

```
1
2
3
4
6
7
8
9
10
```

- **Goto Statement :**

The C++ goto statement is also known as jump statement. It is used to transfer control to the other part of the program. It unconditionally jumps to the specified label.

It can be used to transfer control from deeply nested loop or switch case label.

Example

Let's see the simple example of goto statement in C++.

```
#include <iostream>
using namespace std;
int main()
{
ineligible:
       cout<<"You are not eligible to vote!\n";
      cout<<"Enter your age:\n";
      int age;
      cin>>age;
      if (age < 18){
            goto ineligible;
      }
      else
      {
            cout<<"You are eligible to vote!";
      }
  }
```
Output:
```
You are not eligible to vote!
Enter your age:
16
You are not eligible to vote!
Enter your age:
7
You are not eligible to vote!
Enter your age:
22
You are eligible to vote!
```

- **C++ Functions :**

The function in C++ language is also known as procedure or subroutine in other programming languages.

To perform any task, we can create function. A function can be called many times. It provides modularity and code reusability.

➢ **Advantage of functions in C++ :**

There are many advantages of functions.

1) **Code Reusability**

   By creating functions in C++, you can call it many times. So we don't need to write the same code again and again.

2) **Code optimization**

   It makes the code optimized, we don't need to write much code. Suppose, you have to check 3 numbers (531, 883 and 781) whether it is prime number or not. Without using function, you need to write the prime number logic 3 times. So, there is repetition of code. But if you use functions, you need to write the logic only once and you can reuse it several times.

➢ **Types of Functions :**

There are two types of functions in C ++ programming:

1. Library Functions: are the functions which are declared in the C++ header files such as ceil(x), cos(x), exp(x), etc.

2. User-defined functions: are the functions which are created by the C++ programmer, so that he/she can use it many times. It reduces complexity of a big program and optimizes the code.

➢ **Declaration of a function :**

The syntax of creating function in C++ language is given below:

```
return_type function_name(data_type parameter...)
{
//code to be executed
}
```

Example
Let's see the simple example of C++ function.

```
#include <iostream>
using namespace std;
void func() {
   static int i=0; //static variable
   int j=0; //local variable
   i++;
   j++;
   cout<<"i=" << i<<" and j=" <<j<<endl;
}
int main()
{
 func();
 func();
 func();
}
```

Output:

```
i= 1 and j= 1
i= 2 and j= 1
i= 3 and j= 1
```

- **Call by value and call by reference in C++ :**

There are two ways to pass value or data to function in C language: call by value and call by reference. Original value is not modified in call by value but it is modified in call by reference.

➢ **Call by value in C++ :**

In call by value, original value is not modified.

In call by value, value being passed to the function is locally stored by the function parameter in stack memory location. If you change the value of function parameter, it is changed for the current function only. It will not change the value of variable inside the caller method such as main().

Example

```
#include <iostream>
using namespace std;
void change(int data);
int main()
{
int data = 3;
change(data);
cout << "Value of the data is: " << data<< endl;
return 0;
}
void change(int data)
{
data = 5;
}
```

Output:

```
Value of the data is: 3
```

- **Call by reference in C++ :**

In call by reference, original value is modified because we pass reference (address).

Here, address of the value is passed in the function, so actual and formal arguments share the same address space. Hence, value changed inside the function, is reflected inside as well as outside the function.

Example

```
#include<iostream>
using namespace std;
void swap(int *x, int *y)
{
 int swap;
 swap=*x;
 *x=*y;
 *y=swap;
}
int main()
{
 int x=500, y=100;
 swap(&x, &y);  // passing value to function
 cout<<"Value of x is: "<<x<<endl;
 cout<<"Value of y is: "<<y<<endl;
 return 0;
}
```

Output:

```
Value of x is: 100
Value of y is: 500
```

- **C++ Recursion :**

When function is called within the same function, it is known as recursion in C++. The function which calls the same function, is known as recursive function.

A function that calls itself, and doesn't perform any task after function call, is known as tail recursion. In tail recursion, we generally call the same function with return statement.

Let's see a simple example of recursion.

```
recursionfunction(){
recursionfunction(); //calling self function
}
```

Example

```
#include<iostream>
using namespace std;
int main()
{
int factorial(int);
int fact,value;
cout<<"Enter any number: ";
cin>>value;
fact=factorial(value);
cout<<"Factorial of a number is: "<<fact<<endl;
return 0;
}
int factorial(int n)
{
if(n<0)
return(-1); /*Wrong value*/
if(n==0)
return(1);  /*Terminating condition*/
else
{
return(n*factorial(n-1));
}
}
```

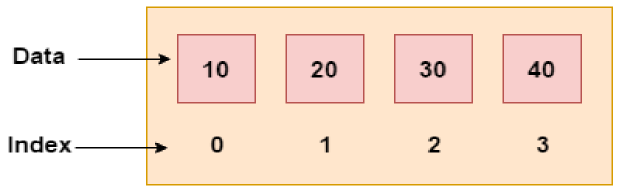
Output:

```
Enter any number: 5
Factorial of a number is: 120
```

- **C++ Arrays :**

Like other programming languages, array in C++ is a group of similar types of elements that have contiguous memory location.

In C++ std::array is a container that encapsulates fixed size arrays. In C++, array index starts from 0. We can store only fixed set of elements in C++ array.



➢ **Advantages of C++ Array :**

1. Code Optimization (less code)
2. Random Access
3. Easy to traverse data
4. Easy to manipulate data
5. Easy to sort data etc.

➢ **Disadvantages of C++ Array :**

1. Fixed size

- **C++ Array Types :**

There are 2 types of arrays in C++ programming:

1. Single Dimensional Array
2. Multidimensional Array

➢ **C++ Single Dimensional Array**

Example of C++ array, where we are going to create, initialize and traverse array.

```
#include <iostream>
using namespace std;
int main()
{
 int arr[5]={10, 0, 20, 0, 30};  //creating and initializing array
        //traversing array
        for (int i = 0; i < 5; i++)
        {
            cout<<arr[i]<<"\n";
        }
    }
```

Output:

```
10
0
20
0
30
```

- **C++ Passing Array to Function**

In C++, to reuse the array logic, we can create function. To pass array to function in C++, we need to provide only array name.

functionname(arrayname); //passing array to function

Example: print array elements

```
#include <iostream>
using namespace std;
void printArray(int arr[5]);
int main()
{
      int arr1[5] = { 10, 20, 30, 40, 50 };
      int arr2[5] = { 5, 15, 25, 35, 45 };
      printArray(arr1); //passing array to function
      printArray(arr2);
}
void printArray(int arr[5])
{
   cout << "Printing array elements:"<< endl;
   for (int i = 0; i < 5; i++)
   {
              cout<<arr[i]<<"\n";
   }
}
```

Output:

```
Printing array elements:
10
20
30
40
50
```

```
Printing array elements:
5
15
25
35
45
```

- **C++ Multidimensional Arrays :**

The multidimensional array is also known as rectangular arrays in C++. It can be two dimensional or three dimensional. The data is stored in tabular form (row ∗ column) which is also known as matrix.

Example

Example of multidimensional array in C++ which declares, initializes and traverse two dimensional arrays.

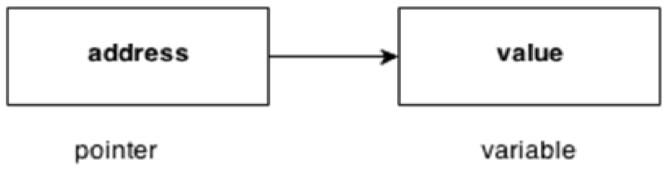
```
#include <iostream>
using namespace std;
int main()
{
  int test[3][3];  //declaration of 2D array
    test[0][0]=5;  //initialization
    test[0][1]=10;
    test[1][1]=15;
    test[1][2]=20;
    test[2][0]=30;
    test[2][2]=10;
    //traversal
    for(int i = 0; i < 3; ++i)
    {
        for(int j = 0; j < 3; ++j)
        {
            cout<< test[i][j]<<" ";
        }
        cout<<"\n"; //new line at each row
    }
    return 0;
}
```

Output:

```
5 10 0
0 15 20
30 0 10
```

- ## C++ Pointers :

The pointer in C++ language is a variable, it is also known as locator or indicator that points to an address of a value.



Example : Pointer Program to swap 2 numbers without using 3rd variable

```
#include <iostream>
using namespace std;
int main()
{
int a=20,b=10,∗p1=&a,∗p2=&b;
cout<<"Before swap: ∗p1="<<∗p1<<" ∗p2="<<∗p2<<endl;
∗p1=∗p1+∗p2;
∗p2=∗p1-∗p2;
∗p1=∗p1-∗p2;
cout<<"After swap: ∗p1="<<∗p1<<" ∗p2="<<∗p2<<endl;
  return 0;
}
```

Output:

```
Before swap: ∗p1=20 ∗p2=10
After swap: ∗p1=10 ∗p2=20
```

Programs :

- **C++ Program to print number entered by user.**

```
#include <iostream>
using namespace std;
int main()
{
  int number ;
  cout<< " Enter an integer :" ;
  cin>>number;
  cout<< "you entered" << number;
  return 0;
}
```

Output : Enter an integer : 23

```
you entered 23
```

- **C++ Program to add two numbers.**

```
include <iostream>
using namespace std;

int main() {

  int first_number, second_number, sum;

  cout << "Enter two integers: ";
  cin >> first_number >> second_number;

  // sum of two numbers in stored in variable sumOfTwoNumbers
  sum = first_number + second_number;

  // prints sum
  cout << first_number << " + " <<  second_number << " = " << sum;

  return 0;
}
```

Output :

```
Enter two integers : 4
                     5
4+5 = 9
```

- **C++ Program to find Quotient & Remainder.**

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

int main()
{
    int divisor, dividend, quotient, remainder;

    cout << "Enter dividend: ";
    cin >> dividend;

    cout << "Enter divisor: ";
    cin >> divisor;

    quotient = dividend / divisor;
    remainder = dividend % divisor;
    cout << "Quotient = " << quotient << endl;
    cout << "Remainder = " << remainder;

    return 0;
}
```
Output :

```
Enter dividend: 13
Enter divisor: 4
Quotient = 3
Remainder = 1
```

- **C++ Program to swap two numbers.**

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

int main()
{
    int a = 5, b = 10, temp;

    cout << "Before swapping." << endl;
    cout << "a = " << a << ", b = " << b << endl;

    temp = a;
    a = b;
    b = temp;

    cout << "\nAfter swapping." << endl;
    cout << "a = " << a << ", b = " << b << endl;

    return 0;
}
```

Output :

```
Before swapping.
a = 5, b = 10

After swapping.
a = 10, b = 5
```

- **C++ Program to print inverted half pyramid using alphabets.**

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

int main()
{
    char input, alphabet = 'A';

    cout << "Enter the uppercase character you want to print in the last row: ";
    cin >> input;

    for(int i = 1; i <= (input-'A'+1); ++i)
    {
        for(int j = 1; j <= i; ++j)
        {
            cout << alphabet << " ";
        }
        ++alphabet;

        cout << endl;
    }
    return 0;
}
```

Output :

```
A
B B
C C C
D D D D
E E E E E
```

- **C++ Program to check whether number is even or odd.**

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

int main() {
  int n;

  cout << "Enter an integer: ";
  cin >> n;

  if ( n % 2 == 0)
    cout << n << " is even.";
  else
```

```
        cout << n << " is odd.";

      return 0;
    }
```

Output :

```
Enter an integer: 23
23 is odd.
```

- **C++ Program to find largest number among three numbers.**

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

int main() {
    float n1, n2, n3;

    cout << "Enter three numbers: ";
    cin >> n1 >> n2 >> n3;

    if(n1 >= n2 && n1 >= n3)
        cout << "Largest number: " << n1;

    if(n2 >= n1 && n2 >= n3)
        cout << "Largest number: " << n2;

    if(n3 >= n1 && n3 >= n2)
        cout << "Largest number: " << n3;

    return 0;
}
```

Output :

```
Enter three numbers: 2.3
8.3
-4.2
Largest number: 8.3
```

- **C++ Program to print Fibonacci series using recursion function.**

```
#include<iostream>
using namespace std;
void printFibonacci(int n){
    static int n1=0, n2=1, n3;
    if(n>0){
         n3 = n1 + n2;
```

```
            n1 = n2;
            n2 = n3;
    cout<<n3<<" ";
            printFibonacci(n-1);
        }
   }
   int main(){
       int n;
       cout<<"Enter the number of elements: ";
       cin>>n;
       cout<<"Fibonacci Series: ";
       cout<<"0 "<<"1 ";
       printFibonacci(n-2);  //n-2 because 2 numbers are already printed
        return 0;
}
```

Output :

```
Enter the number of elements: 15
Fibonacci Series: 0 1 1 2 3 5 8 13 21 34 55 89 144 233 377
```

- **C++ Program to print Fibonacci series without recursion function.**

```
#include <iostream>
using namespace std;
int main() {
  int n1=0,n2=1,n3,i,number;
 cout<<"Enter the number of elements: ";
 cin>>number;
 cout<<n1<<" "<<n2<<" "; //printing 0 and 1
 for(i=2;i<number;++i) //loop starts from 2 because 0 and 1 are already printed
 {
  n3=n1+n2;
  cout<<n3<<" ";
  n1=n2;
  n2=n3;
 }
   return 0;
   }
```

Output :

```
Enter the number of elements: 10
0 1 1 2 3 5 8 13 21 34
```

- **C++ Program to calculate sum of natural numbers.**

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

int main() {
   int n, sum = 0;

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

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

   cout << "Sum = " << sum;
   return 0;
}
```

Output :

```
Enter a positive integer: 50
Sum = 1275
```

- **C++ Program to find factorial of a given number.**

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

int main() {
   int n;
   long factorial = 1.0;

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

   if (n < 0)
      cout << "Error! Factorial of a negative number doesn't exist.";
   else {
      for(int i = 1; i <= n; ++i) {
         factorial *= i;
      }
      cout << "Factorial of " << n << " = " << factorial;
   }

   return 0;
}
```

Output :

```
Enter a positive integer: 4
Factorial of 4 = 24
```

- **C++ Program to generate multiplication table.**

```
#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;
}
```

Output :

```
Enter an integer: 5
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
5 * 6 = 30
5 * 7 = 35
5 * 8 = 40
5 * 9 = 45
5 * 10 = 50
```

- **C++ Program to reverse a number.**

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

int main() {

  int n, reversed_number = 0, remainder;
```

```
    cout << "Enter an integer: ";
    cin >> n;

    while(n != 0) {
      remainder = n % 10;
      reversed_number = reversed_number * 10 + remainder;
      n /= 10;
    }

    cout << "Reversed Number = " << reversed_number;

    return 0;
  }
```

Output :

```
Enter an integer: 2345
Reversed number = 5432
```

- **C++ Program to find all roots of Quadratic equations**.

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

int main() {

    float a, b, c, x1, x2, discriminant, realPart, imaginaryPart;
    cout << "Enter coefficients a, b and c: ";
    cin >> a >> b >> c;
    discriminant = b*b - 4*a*c;

    if (discriminant > 0) {
        x1 = (-b + sqrt(discriminant)) / (2*a);
        x2 = (-b - sqrt(discriminant)) / (2*a);
        cout << "Roots are real and different." << endl;
        cout << "x1 = " << x1 << endl;
        cout << "x2 = " << x2 << endl;
    }

    else if (discriminant == 0) {
        cout << "Roots are real and same." << endl;
        x1 = -b/(2*a);
        cout << "x1 = x2 =" << x1 << endl;
    }

    else {
        realPart = -b/(2*a);
        imaginaryPart =sqrt(-discriminant)/(2*a);
        cout << "Roots are complex and different."  << endl;
        cout << "x1 = " << realPart << "+" << imaginaryPart << "i" << endl;
        cout << "x2 = " << realPart << "-" << imaginaryPart << "i" << endl;
    }

    return 0;
}
```

Output :

```
Enter coefficients a, b and c: 4
5
1
Roots are real and different.
x1 = -0.25
x2 = -1
```