

# Programming in C/C++

## **UNIT-I**

C language: Introduction ; Tokens; Keywords; Identifier ; Variables; Constants; Operators ; Expression; Data types; Operator precedence

Statement: Input statement, Output statement, Conditional and Unconditional Control Statement – Looping Statement: while, do-while, for – nested loop – Arrays.

Overview of C++: Object oriented programming, Introducing C++ classes, Concepts of object oriented programming. Classes & Objects : Classes, Structure & classes, Union & Classes, Friend function, Friend classes, Inline function, Scope resolution operator, Static class members: Static data member, Static member function, Passing objects to function, Returning objects, Object assignment.

## **UNIT-II**

Array and Pointers references: Array of objects, Pointers to object, Type checking C++ pointers, The This pointer, Pointer to derived types, Pointer to class members, References: Reference parameter, Passing references to objects, Returning reference, Independent reference, C++ 's dynamic allocation operators, Initializing allocated memory, Allocating Array, Allocating objects.

Constructor & Destructor: Introduction, Constructor, Parameterized constructor, Multiple constructor in a class, Constructor with default argument, Copy constructor, Default Argument, Constructing two dimensional Array, Destructor.

### **UNIT-III**

Function & operator overloading : Function overloading, Overloading constructor function finding the address of an overloaded function, Operator Overloading: Creating a member operator function, Creating Prefix & Postfix forms of the increment & decrement operation, Overloading the shorthand operation (i.e. +=, -= etc), Operator overloading restrictions, Operator overloading using friend function, Overloading New & Delete, Overloading some special operators, Overloading [ ], ( ), -, comma operator, Overloading << .

### **UNIT-IV**

Inheritance : Base class Access control, Inheritance & protected members, Protected base class inheritance, Inheriting multiple base classes, Constructors, destructors & Inheritance, When constructor & destructor function are executed, Passing parameters to base class constructors, Granting access. Virtual functions & Polymorphism: Virtual base classes; Virtual function, Pure Virtual functions, early Vs. late binding

### **UNIT-V**

String Handling: String declaration; String library functions; String Manipulation; Creating string objects, manipulating string objects, relational operators, string characteristics, Comparing and swapping  
Sorting: Bubble sort, Selection sort, Insertion sort  
Searching: Linear search, Binary search

## **What is C?**

C is a general-purpose programming language created by Dennis Ritchie at the Bell Laboratories in 1972.

It is a very popular language, despite being old.

C is strongly associated with UNIX, as it was developed to write the UNIX operating system.

## **Why Learn C?**

It is one of the most popular programming language in the world

If you know C, you will have no problem learning other popular programming languages such as Java, Python, C++, C#, etc, as the syntax is similar

C is very fast, compared to other programming languages, like Java and Python

C is very versatile; it can be used in both applications and technologies

## **Similarities between C and C++ are:**

Both the languages have a similar syntax.

Code structure of both the languages are same.

The compilation of both the languages is similar.

They share the same basic syntax. Nearly all of C's operators and keywords are also present in C++ and do the same thing.

C++ has a slightly extended grammar than C, but the basic grammar is the same.

Basic memory model of both is very close to the hardware.

Same notions of stack, heap, file-scope and static variables are present in both the languages.

## C

C was developed by Dennis Ritchie between the year 1969 and 1973 at AT&T Bell Labs.

C does not support polymorphism, encapsulation, and inheritance which means that C does not support object oriented programming.

C is a subset of C++.

C contains 32 [keywords](#).

For the development of code, C supports [procedural programming](#).

Data and functions are separated in C because it is a procedural programming language.

## C++

C++ was developed by Bjarne Stroustrup in 1979.

C++ supports [polymorphism](#), [encapsulation](#), and [inheritance](#) because it is an object oriented programming language.

C++ is a superset of C.

C++ contains 63 [keywords](#).

C++ is known as hybrid language because C++ supports both [procedural](#) and [object oriented programming paradigms](#).

Data and functions are encapsulated together in form of an object in C++

# C

C does not support information hiding.

Built-in data types is supported in C.

C is a function driven language because C is a procedural programming language.

Function and operator overloading is not supported in C.

C is a function-driven language.

Functions in C are not defined inside structures.

Namespace features are not present inside the C.

Header file used by C is [stdio.h](#).

Reference variables are not supported by C.

Virtual and friend functions are not supported by C.

C does not support inheritance.

# C++

Data is hidden by the Encapsulation to ensure that data structures and operators are used as intended.

Built-in & user-defined data types is supported in C++.

C++ is an object driven language because it is an object oriented programming.

Function and operator overloading is supported by C++.

C++ is an object-driven language

Functions can be used inside a structure in C++.

[Namespace](#) is used by C++, which avoid name collisions.

Header file used by C++ is [iostream.h](#).

Reference variables are supported by C++.

[Virtual](#) and [friend functions](#) are supported by C++.

C++ supports inheritance.

## **Applications of C Programming**

C was initially used for system development work, particularly the programs that make-up the operating system. C was adopted as a system development language because it produces code that runs nearly as fast as the code written in assembly language. Some examples of the use of C are -

Operating Systems

Language Compilers

Assemblers

Text Editors

Print Spoolers

Network Drivers

Modern Programs

Databases

Language Interpreters

Utilities



A C program basically consists of the following parts –

Preprocessor Commands

Functions

Variables

Statements & Expressions

Comments

Let us look at a simple code that would print the words "Hello World" –

```
#include <stdio.h>
```

```
int main() {  
    /* my first program in C */  
    printf("Hello, World! \n");  
  
    return 0;  
}
```

## C++ Program

Before starting the abcd of C++ language, we need to learn how to write, compile and run the first C++ program.

To write the first C++ program, open the C++ console and write the following code:

```
#include <iostream.h>
```

```
#include<conio.h>
```

```
void main() {
```

```
    clrscr();
```

```
    cout << "Welcome to C++ Programming.";
```

```
    getch();
```

```
}
```

`#include<iostream.h>` includes the standard input output library functions. It provides cin and cout methods for reading from input and writing to output respectively.

`#include <conio.h>` includes the console input output library functions. The `getch()` function is defined in `conio.h` file.

`void main()` The `main()` function is the entry point of every program in C++ language. The `void` keyword specifies that it returns no value.

`cout << "Welcome to C++ Programming."` is used to print the data "Welcome to C++ Programming." on the console.

`getch()` The `getch()` function asks for a single character. Until you press any key, it blocks the screen.

## How to compile and run the C++ program

There are 2 ways to compile and run the C++ program, by menu and by shortcut.

### By menu

Now **click on the compile menu then compile sub menu** to compile the c++ program.

Then **click on the run menu then run sub menu** to run the c++ program.

### By shortcut

**Or, press ctrl+f9** keys compile and run the program directly.

You will see the following output on user screen.

we can view the user screen any time by pressing the **alt+f5** keys.

Now **press Esc** to return to the turbo c++ console.

## C++ Basic Input/Output

C++ I/O operation is using the stream concept. Stream is the sequence of bytes or flow of data. It makes the performance fast.

If bytes flow from main memory to device like printer, display screen, or a network connection, etc, this is called as **output operation**.

If bytes flow from device like printer, display screen, or a network connection, etc to main memory, this is called as **input operation**.

## I/O Library Header Files

Let us see the common header files used in C++ programming are

Header File	Function and Description
<iostream>	It is used to define the cout, cin and cerr objects, which correspond to standard output stream, standard input stream and standard error stream, respectively.
<iomanip>	It is used to declare services useful for performing formatted I/O, such as setprecision and setw.
<fstream>	It is used to declare services for user-controlled file processing.

## Standard output stream (cout)

The **cout** is a predefined object of **ostream** class. It is connected with the standard output device, which is usually a display screen. The cout is used in conjunction with stream insertion operator (<<) to display the output on a console

Let's see the simple example of standard output stream (cout):

```
include <iostream>
using namespace std;
int main( ) {
    char ary[] = "Welcome to C++ tutorial";
    cout << "Value of ary is: " << ary << endl;
}
```

## Standard input stream (cin)

The cin is a predefined object of istream class. It is connected with the standard input device, which is usually a keyboard. The cin is used in conjunction with stream extraction operator (>>) to read the input from a console.

Let's see the simple example of standard input stream (cin):

```
#include <iostream.h>
int main( ) {
    int age;
    cout << "Enter your age: ";
    cin >> age;
    cout << "Your age is: " << age << endl;
}
```

### Standard end line (endl)

The endl is a predefined object of ostream class. It is used to insert a new line characters and flushes the stream.

Let's see the simple example of standard end line (endl):

```
#include <iostream.h>
int main( ) {
    cout << "C++ Tutorial";
    cout << " Javatpoint"<<endl;
    cout << "End of line"<<endl;
}
```

## C++ Variable

A variable is a name of memory location. It is used to store data. Its value can be changed and it can be reused many times.

It is a way to represent memory location through symbol so that it can be easily identified.

Let's see the syntax to declare a variable:

```
type variable_list;
```

```
int x;
```

```
float y;
```

```
char z;
```

Here, x, y, z are variables and int, float, char are data types.

We can also provide values while declaring the variables as given below:

```
int x=5,b=10; //declaring 2 variable of integer type
```

```
float f=30.8;
```

```
char c='A';
```

## **Rules for defining variables**

A variable can have alphabets, digits and underscore.

A variable name can start with alphabet and underscore only. It can't start with digit.

No white space is allowed within variable name.

A variable name must not be any reserved word or keyword  
e.g. char, float etc.

Valid variable names:

```
int a;  
int _ab;  
int a30;
```

Invalid variable names:

```
int 4;  
int x y;  
int double;
```



## C++ Data Types

A data type specifies the type of data that a variable can store such as integer, floating, character etc.

There are 4 types of data types in C++ language.

### Types

Basic Data Type

Derived Data Type

Enumeration Data Type

User Defined Data Type

### Data Types

int, char, float, double, etc

array, pointer, etc

enum

structure

## Basic Data Types

The basic data types are integer-based and floating-point based. C++ language supports both signed and unsigned literals.

The memory size of basic data types may change according to 32 or 64 bit operating system.

## C++ Keywords

A keyword is a reserved word. You cannot use it as a variable name, constant name etc. A list of 32 Keywords in C++ Language which are also available in C language are given below.

auto	break	case	char	const	continue	
	default	do				
double	else	enum	extern	float	for	goto
	if					
int	long	register	return	short	signed	
	sizeof	static				
struct	switch	typedef	union	unsigned	void	
	volatile	while				

## **C++ Operators**

An operator is simply a symbol that is used to perform operations. There can be many types of operations like arithmetic, logical, bitwise etc.

There are following types of operators to perform different types of operations in C language.

Arithmetic Operators

Relational Operators

Logical Operators

Bitwise Operators

Assignment Operator

Unary operator

Ternary or Conditional Operator

Misc Operator

	Operator	Type
<b>Binary Operator</b>	+, -, *, /, %	Arithmetic Operators
	<, <=, >, >=, ==, !=	Relational Operators
	&&,   , !	Logical Operators
	&,  , <<, >>, ~, ^	Bitwise Operators
	=, +=, -=, *=, /=, %=	Assignment Operators
<b>Unary Operator</b>	→ ++, --	Unary Operator
<b>Ternary Operator</b>	→ ?:	Ternary or Conditional Operator

## **C++ Identifiers**

C++ identifiers in a program are used to refer to the name of the variables, functions, arrays, or other user-defined data types created by the programmer. They are the basic requirement of any language. Every language has its own rules for naming the identifiers.

In short, we can say that the C++ identifiers represent the essential elements in a program which are given below:

Constants

Variables

Functions

Labels

Defined data types

Some naming rules are common in both C and C++. They are as follows:

1. Only alphabetic characters, digits, and underscores are allowed.
2. The identifier name cannot start with a digit, i.e., the first letter should be alphabetical. After the first letter, we can use letters, digits, or underscores.
3. In C++, uppercase and lowercase letters are distinct. Therefore, we can say that C++ identifiers are case-sensitive.
4. A declared keyword cannot be used as a variable name.

For example, suppose we have two identifiers, named as 'FirstName', and 'Firstname'. Both the identifiers will be different as the letter 'N' in the first case is in uppercase while lowercase in second. Therefore, it proves that identifiers are case-sensitive.

## Valid Identifiers

The following are the examples of valid identifiers are:

Result

Test2

\_sum

power

## Invalid Identifiers

The following are the examples of invalid identifiers:

Sum-1 // containing special character '-'.  
2data // the first letter is a digit.  
break // use of a keyword



**Let's look at a simple example to understand the concept of identifiers.**

```
#include <iostream.h>
```

```
int main()
{
    int a;
    int A;
    cout<<"Enter the values of 'a' and 'A'";
    cin>>a;
    cin>>A;
    cout<<"\nThe values that you have entered are : "<<a<<" , "<<A;
    return 0;
}
```

In the above code, we declare two variables 'a' and 'A'. Both the letters are same but they will behave as different identifiers. As we know that the identifiers are the case-sensitive so both the identifiers will have different memory locations.

### **What are the keywords?**

Keywords are the reserved words that have a special meaning to the compiler. They are reserved for a special purpose, which cannot be used as the identifiers. For example, 'for', 'break', 'while', 'if', 'else', etc. are the predefined words where predefined words are those words whose meaning is already known by the compiler. Whereas, the identifiers are the names which are defined by the programmer to the program elements such as variables, functions, arrays, objects, classes.

## C++ if-else

In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

if statement

if-else statement

nested if statement

if-else-if ladder

## C++ if-else

In C++ programming, if statement is used to test the condition. There are various types of if statements in C++.

if statement

if-else statement

nested if statement

if-else-if ladder

### C++ IF Statement

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

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

### C++ If Example

```
#include <iostream.h>  
int main () {  
    int num = 10;  
    if (num % 2 == 0)  
    {  
        cout<<"It is even number";  
    }  
    return 0;  
}
```

## C++ 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  
}
```

### C++ If-else Example

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

### C++ If-else Example: with input from user

```
#include <iostream>  
int main () {  
    int num;  
    cout<<"Enter a Number: ";  
    cin>>num;  
    if (num % 2 == 0)  
    {  
        cout<<"It is even number"<<endl;  
    }  
    else  
    {  
        cout<<"It is odd number"<<endl;  
    }  
    return 0;  
}
```

## C++ 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
}
```

```
#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";
    }
}
```

## C++ switch

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;  
}
```

```
#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;  
    }  
}
```

In programming, sometimes there is a need to perform some operation more than once or (say) n number of times. Loops come into use when we need to repeatedly execute a block of statements.

```
// C++ program to illustrate while loop
// C++ program to illustrate need of loops
```

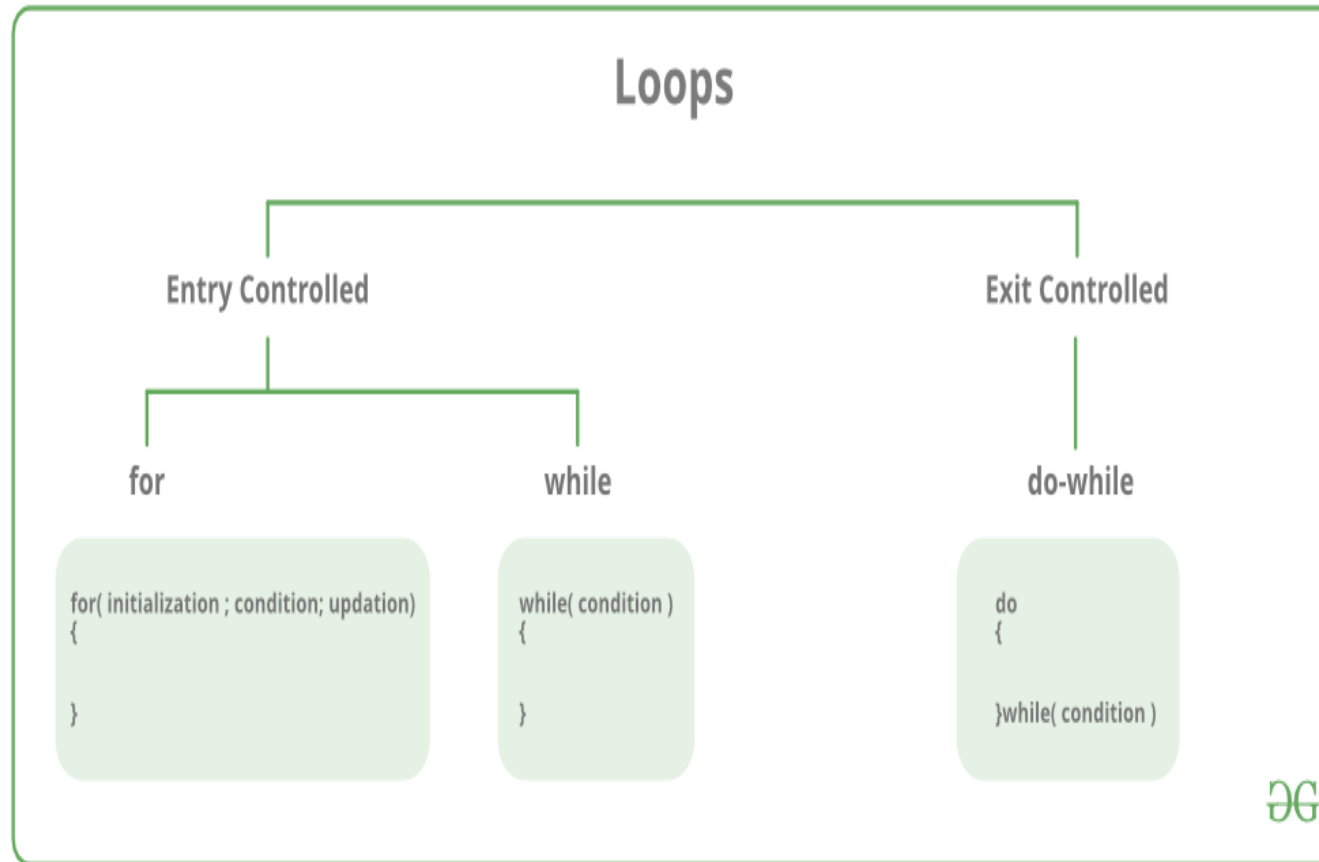
```
// C++ program to illustrate need
of loops
#include <iostream>
```

[illegible]



## Using Loops

In Loop, the statement needs to be written only once and the loop will be executed 10 times as shown below.



## Loop Type and Description

1. while loop – First checks the condition, then executes the body.
2. for loop – firstly initializes, then, condition check, execute body, update.
3. do-while – firstly, execute the body then condition check

## for Loop

A for loop is a repetition control structure that allows us to write a loop that is executed a specific number of times. The loop enables us to perform n number of steps together in one line.

Syntax:

```
for (initialization expr; test expr; update expr)
{
    // body of the loop
    // statements we want to execute
}
```

Example:

```
for(int i = 0; i < n; i++){
}
```

```
// C++ program to illustrate for loop
#include <iostream.h>
```

```
int main()
{
    for (int i = 1; i <= 10; i++)
    {
        cout << "Hello World\n";
    }

    return 0;
}
```

## C++ Functions

```
#include<iostream.h>
#include<conio.h>

void add(int a, int b);
void main()
{
    int a=5,b=6;
    clrscr();
    add(a,b);
    getch();
}
void add(int c, int d)
{
    int f;
    f=c+d;
    cout<<f;
}
```

A function is a block of code which only runs when it is called.

You can pass data, known as parameters, into a function.

Functions are used to perform certain actions, and they are important for reusing code: Define the code once, and use it many times.

### Create a Function

C++ provides some pre-defined functions, such as `main()`, which is used to execute code. But you can also create your own functions to perform certain actions.

To create (often referred to as declare) a function, specify the name of the function, followed by parentheses `()`:

Main differences between the structure and class

Here, we are going to discuss the main differences between the structure and class. Some of them are as follows:

By default, all the members of the structure are public. In contrast, all members of the class are private.

The structure will automatically initialize its members. In contrast, constructors and destructors are used to initialize the class members.

When a structure is implemented, memory allocates on a stack. In contrast, memory is allocated on the heap in class.

Variables in a structure cannot be initialized during the declaration, but they can be done in a class.

There can be no null values in any structure member. On the other hand, the class variables may have null values.

A structure is a value type, while a class is a reference type.

Operators to work on the new data form can be described using a special method.

## C++ OOPs Concepts

The major purpose of C++ programming is to introduce the concept of object orientation to the C programming language.

Object Oriented Programming is a paradigm that provides many concepts such as inheritance, data binding, polymorphism etc.

The programming paradigm where everything is represented as an object is known as truly object-oriented programming language. Smalltalk is considered as the first truly object-oriented programming language.

## OOPs (Object Oriented Programming System)

Object means a real word entity such as pen, chair, table etc. Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance by providing some concepts:

## Object

Any entity that has state and behavior is known as an object. For example: chair, pen, table, keyboard, bike etc. It can be physical and logical.

## Class

Collection of objects is called class. It is a logical entity.

## Inheritance

When one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.

## Polymorphism

When one task is performed by different ways i.e. known as polymorphism. For example: to convince the customer differently, to draw something e.g. shape or rectangle etc.

In C++, we use Function overloading and Function overriding to achieve polymorphism.

## Abstraction

Hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing.

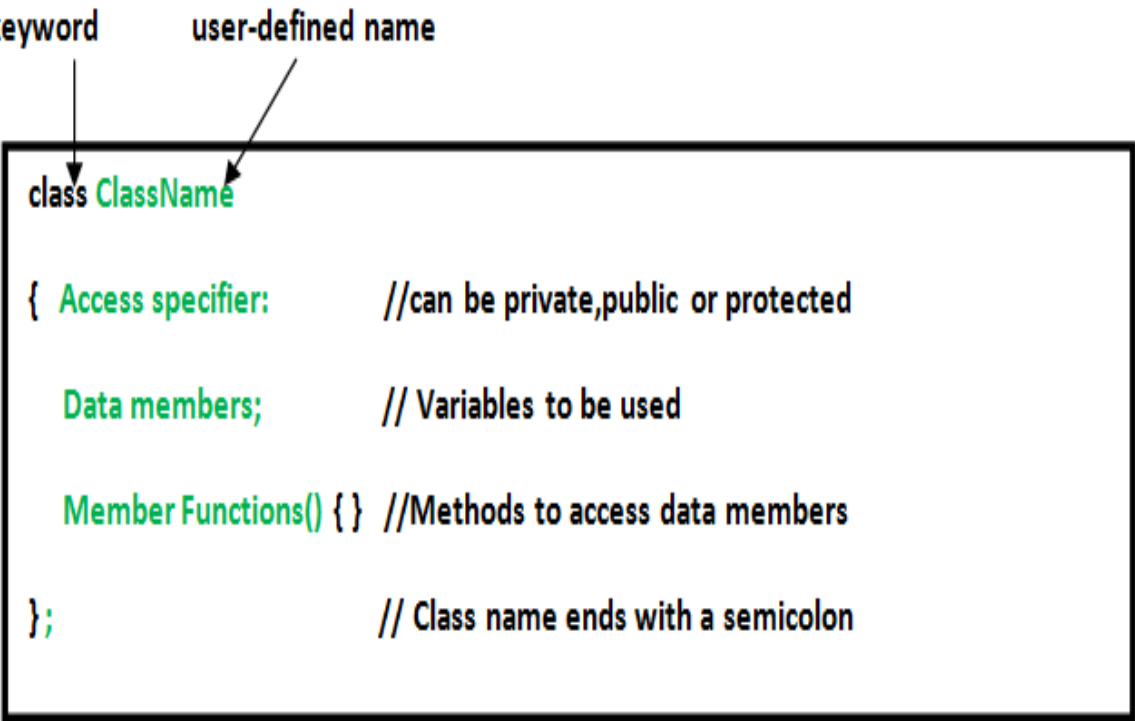
In C++, we use abstract class and interface to achieve abstraction.

## Encapsulation

Binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

keyword

user-defined name



```
class ClassName  
  
{ Access specifier:      //can be private,public or protected  
  
  Data members;        // Variables to be used  
  
  Member Functions() { } //Methods to access data members  
  
};                      // Class name ends with a semicolon
```

The diagram illustrates the syntax of a C++ class definition. It is enclosed in a black rectangular box. Above the box, two labels with arrows point to specific parts of the code: 'keyword' points to the word 'class', and 'user-defined name' points to 'ClassName'. The code itself is as follows:

- `class ClassName`: The first line, where `class` is the keyword and `ClassName` is the user-defined name.
- `{ Access specifier: //can be private,public or protected`: The opening curly brace followed by an access specifier and a comment.
- `Data members; // Variables to be used`: A line for data members with a comment.
- `Member Functions() { } //Methods to access data members`: A line for member functions with a comment.
- `}; // Class name ends with a semicolon`: The closing curly brace followed by a semicolon and a comment.



## C++ Object and Class Example

```
#include <iostream.h>

class Student {
public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
};

int main() {
    Student s1; //creating an object of Student
    s1.id = 201;
    s1.name = "Mtech";
    cout<<s1.id<<endl;
    cout<<s1.name<<endl;
    return 0;
}
```

## C++ Class Example: Initialize and Display data through method

```
#include <iostream>
using namespace std;
class Student {
public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable)
    void insert(int i, string n)
    {
        id = i;
        name = n;
    }
    void display()
    {
        cout<<id<<" "<<name<<endl;
    }
};

int main(void) {
    Student s1; //creating an object of Student
    Student s2; //creating an object of Student
    s1.insert(201, "Mtech");
    s2.insert(202, "MSC");
    s1.display();
    s2.display();
    return 0;
}
```

```
#include<iostream.h>
#include<conio.h>
class student
{
    private:

        char name[20],regd[10],branch[10];
        int sem;
    public:
        void input();
        void display();

};
void student::input()
{
    cout<<"Enter Name:";
    cin>>name;
    cout<<"Enter Regdno.:";
    cin>>regd;
    cout<<"Enter Branch:";
    cin>>branch;
    cout<<"Enter Sem:";
    cin>>sem;
}
void student::display()
{
    cout<<"\nName:"<<name;
    cout<<"\nRegdno.:"<<regd;
    cout<<"\nBranch:"<<branch;
    cout<<"\nSem:"<<sem;
}
int main()
{
    student s;
    s.input();
    s.display();
    getch();
}
```

Class	Structure
Members of a class are private by default.	Members of a structure are public by default.
Member classes/structures of a class are private by default.	Member classes/structures of a structure are public by default.
It is declared using the <b>class</b> keyword.	It is declared using the <b>struct</b> keyword.
It is normally used for data abstraction and further inheritance.	It is normally used for the grouping of data

# Inline Functions

inline function is a function that is expanded in line when it is called. When the inline function is called whole code of the inline function gets inserted or substituted at the point of inline function call. This substitution is performed by the C++ compiler at compile time. Inline function may increase efficiency if it is small.

The syntax for defining the function inline is:

```
inline return-type function-name(parameters)
{
    // function code
}
```

```
#include<iostream.h>
#include<conio.h>
```

```
inline int cube(int s)
{
    return s*s*s;
}
int main()
{
    cout << "The cube of 3 is: " << cube(3) << "\n";
    return 0;
}
```

## Scope resolution operator in C++

The scope resolution operator is used to reference the global variable or member function that is out of scope. Therefore, we use the scope resolution operator to access the hidden variable or function of a program. The operator is represented as the double colon (::) symbol.

```
#include <iostream.h>
#include<conio.h>
// declare global variable
int num = 50;
int main ()
{
// declare local variable
int num = 100;

// print the value of the variables
cout << " The value of the local variable num: " << num;

// use scope resolution operator (::) to access the global
variable
cout << "\n The value of the global variable num: " <<
::num;
return 0;
}
```

// C++ program to show that scope resolution operator :: is used  
// to define a function outside a class

```
#include<iostream.h>
#include<conio.h>

class A
{
public:

// Only declaration
void fun();
};

// Definition outside class using ::
void A::fun()
{
    cout << "fun() called";
}

int main()
{
    A a;
    a.fun();
    return 0;
}
```

Static is a keyword in C and C++ which is used to declare a special type of a variable or a function inside or outside of a class. In this post, we will briefly understand the concept of static member variables and static member functions in c++ and compare them with normal variables and functions in the following order:

Static Member Variables

Static Member Functions in C++

## Static Member Variables

Variables classified as static are also a part of C. suppose in a function there are 2 variables, one is a normal variable and the other one is a static variable. The normal variable is created when the function is called and its scope is limited. While the static variable is created once and destroyed at the end of the program. These variables have a lifetime throughout the program.

```
#include <iostream.h>
```

```
void Test(){
```

```
    static int x = 1;
```

```
    x = ++x;
```

```
    int y = 1;
```

```
    y = ++y;
```

```
    cout<<"x = "<<x<<"n";
```

```
    cout<<"y = "<<y<<"n";
```

```
}
```

```
int main()
```

```
{
```

```
    Test();
```

```
    Test();
```

```
    return 0;
```

```
}
```

```
x = 2
```

```
y = 2
```

```
x = 3
```

```
y = 2
```

```
...Program finished with exit code 0
```

```
Press ENTER to exit console. 
```

From the above output, we can conclude that every time the Test( ) function was called a copy of variable 'y' was created while the same copy of variable 'x' was used every time the Test( ) function was called.

Now, let's discuss the characteristics of the static variables

Static variables are initialized to 0. It is initialized only once.

Throughout the program, only one copy of the static member variable is created for the entire class hence static member variables are also called class variables. It is shared by all instances of the class.

The static member variable is only visible within the class but its lifetime is till the program ends.



Let's consider an example of static member variables in a class.

```
Initial value of x
x = 0
x = 0
x = 0
Value of x after calling function1
x = 3
x = 3
x = 3

...Program finished with exit code 0
Press ENTER to exit console. □
```

```
#include <iostream.h>
class Example{

    static int x;

public:

    void function1(){
        x++;
    }

    void function2(){
        cout<<"x = "<<x<<"n";
    }

};

int Example :: x;

int main()
{
    Example obj1, obj2, obj3;

    cout<<"Initial value of x" <<"n";

    obj1.function2();
    obj2.function2();
    obj3.function2();

    obj1.function1();
    obj2.function1();
    obj3.function1();

    cout<<"Value of x after calling function1"<<"n";

    obj1.function2();
    obj2.function2();
    obj3.function2();

    return 0;
}
```

From the above output, we can see that the variable 'x' is shared across all the objects. To understand the concept of the static data variables in detail we can think of a library where there are several books placed on different shelves. Consider the library as a class, position of a certain book 'x' as a static member variable and students as the objects of the class. When the first student arrived he places 'x' at a new position now when another student arrives 'x' won't return to its original position, but it will remain where the first student left it.

## Static Member Functions in C++

Just like static member variables we have static member functions that are used for a specific purpose. To create a static member function we need to use the static keyword while declaring the function. Since static member variables are class properties and not object properties, to access them we need to use the class name instead of the object name.

Properties of static member functions:

A static function can only access other static variables or functions present in the same class

Static member functions are called using the class name. Syntax- `class_name::function_name( )`

Let's consider a classic example to understand the concept of static member functions in detail. In this example, we will understand all the concepts related to static member functions.

```

#include <iostream>

using namespace std;

class Example{
    static int Number;
    int n;
    public:
    void set_n(){
        n = ++Number;
    }

    void show_n(){
        cout<<"value of n = "<<n<<endl;
    }

    static void show_Number(){
        cout<<"value of Number = "<<Number<<endl;
    }

};

int Example:: Number;

int main()
{
    Example example1, example2;

    example1.set_n();
    example2.set_n();

    example1.show_n();
    example2.show_n();

    Example::show_Number();

    return 0;
}

```

```

Initial value of x
x = 0
x = 0
x = 0
Value of x after calling function1
x = 3
x = 3
x = 3

...Program finished with exit code 0
Press ENTER to exit console.

```

From the above output, we can see that the value of the variable 'n' is different for both the objects 'example1' and 'example2' of the class 'Example'. Since the variable 'Number' is a class variable its value is the same for both the objects 'example1' and 'example2'. Static member variables and functions are used when common values are to be shared across all the objects. While programming, the use of static keyword should be done wisely.

## How to pass and return object from C++ Functions?

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

class Student {

public:
    double marks;

    // constructor to initialize marks
    Student(double m) {
        marks = m;
    }
};

// function that has objects as parameters
void calculateAverage(Student s1, Student s2) {

    // calculate the average of marks of s1 and s2
    double average = (s1.marks + s2.marks) / 2;

    cout << "Average Marks = " << average << endl;

}

int main() {
    Student student1(88.0), student2(56.0);

    // pass the objects as arguments
    calculateAverage(student1, student2);

    return 0;
}
```

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

class Student {
public:
    double marks1, marks2;
};

// function that returns object of Student
Student createStudent() {
    Student student;

    // Initialize member variables of Student
    student.marks1 = 96.5;
    student.marks2 = 75.0;

    // print member variables of Student
    cout << "Marks 1 = " << student.marks1 << endl;
    cout << "Marks 2 = " << student.marks2 << endl;

    return student;
}

int main() {
    Student student1;

    // Call function
    student1 = createStudent();

    return 0;
}
```

## Static Member

We use the static keyword to define the static data member or static member function inside and outside of the class.

Let's understand the static data member and static member function using the programs.

### **Static data member**

When we define the data member of a class using the static keyword, the data members are called the static data member. A static data member is similar to the static member function because the static data can only be accessed using the static data member or static member function. And, all the objects of the class share the same copy of the static member to access the static data.

### Syntax

```
static data_type data_member;
```

Here, the static is a keyword of the predefined library.

```

#include <iostream>
#include <string.h>
using namespace std;
// create class of the Car
class Car
{
private:
int car_id;
char car_name[20];
int marks;

public:
// declare a static data member
static int static_member;

Car()
{
static_member++;
}

void inp()
{
cout << " \n\n Enter the Id of the Car: " << endl;
cin >> car_id; // input the id
cout << " Enter the name of the Car: " << endl;
cin >> car_name;
cout << " Number of the Marks (1 - 10): " << endl;
cin >> marks;
}

// display the entered details
void disp ()
{
cout << " \n Id of the Car: " << car_id;
cout << " \n Name of the Car: " << car_name;
cout << " \n Marks: " << marks;

}
};

```

```

// initialized the static data member to 0
int Car::static_member = 0;

int main ()
{
// create object for the class Car
Car c1;
// call inp() function to insert values
c1. inp ();
c1. disp();

//create another object
Car c2;
// call inp() function to insert values
c2. inp ();
c2. disp();

cout << " \n No. of objects created in the class: " << Car :: static_member << endl;
return 0;
}

```

## Static Member Functions

The static member functions are special functions used to access the static data members or other static member functions.

### Syntax

`class_name::function_name (parameter);`

Here, the `class_name` is the name of the class.

`function_name`: The function name is the name of the static member function.

```
#include <iostream>
using namespace std;
class Note
{
    // declare a static data member
    static int num;

public:
    // create static member function
    static int func ()
    {
        return num;
    }
};
// initialize the static data member using the class name and the
// scope resolution operator
int Note :: num = 5;

int main ()
{
    // access static member function using the class name and the
    // scope resolution
    cout << " The value of the num is: " << Note:: func () << endl;
    return 0;
}
```



## Passing objects to function

```
#include <iostream.h>
class A
{
public:
    int n=100;
    char ch='A';
    void disp(A a)
    {
        cout<<a.n<<endl;
        cout<<a.ch<<endl;
    }
};
int main() {
    A obj;
    obj.disp(obj);
    return 0;
}
```

## Pass object to a function

An object can be passed to a function just like we pass structure to a function. Here in class A we have a function disp() in which we are passing the object of class A. Similarly we can pass the object of another class to a function of different class.

## UNIT-II

Array and Pointers references: Array of objects, Pointers to object, Type checking C++ pointers, The This

pointer, Pointer to derived types, Pointer to class members, References: Reference parameter, Passing

references to objects, Returning reference, Independent reference, C++ 's dynamic allocation operators,

Initializing allocated memory, Allocating Array, Allocating objects

## C++ Arrays

Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.

To declare an array, define the variable type, specify the name of the array followed by square brackets and specify the number of elements it should store:

```

#include<iostream.h>
#include<conio.h>
void main()
{
    int i,j,k,l;
    int arr[2][2];
    clrscr();
    cout<<"enter the element";
    for(i=1;i<=2;i++)
    {
        for(j=1;j<=2;j++)
        {
            cin>>arr[i][j];
        }
    }
    cout<<"The 2*2 array is:" <<"\n";
    for(k=1;k<=2;k++)
    {
        for(l=1;l<=2;l++)
        {
            cout<<arr[k][l]<<"\t";
        }

        cout<<"\n";
    }
    getch();
}

```

```

#include <iostream.h>
#include<conio.h>
void main()
{
    int r, c, a[100][100], b[100][100], sum[100][100], i, j;
    cout << "Enter number of rows (between 1 and 100): ";    cin >> r;
    cout << "Enter number of columns (between 1 and 100): ";    cin >> c;
    cout << endl << "Enter elements of 1st matrix: " << endl;
    // Storing elements of first matrix entered by user.
    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            cout << "Enter element a" << i + 1 << j + 1 << " : ";
            cin >> a[i][j];
        }
    // Storing elements of second matrix entered by user.
    cout << endl << "Enter elements of 2nd matrix: " << endl;
    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
        {
            cout << "Enter element b" << i + 1 << j + 1 << " : ";
            cin >> b[i][j];
        }

    // Adding Two matrices
    for(i = 0; i < r; ++i)
        for(j = 0; j < c; ++j)
            sum[i][j] = a[i][j] + b[i][j];

    // Displaying the resultant sum matrix.
    cout << endl << "Sum of two matrix is: " << endl;
    for(i = 0; i < r; ++i)
    {
        for(j = 0; j < c; ++j)
        {
            cout << sum[i][j] << " ";
        }
        cout<<endl;
    }
    getch();
}

```





In C++, Pointers are variables that hold addresses of other variables. Not only can a pointer store the address of a single variable, it can also store the address of cells of an array.

Consider this example:

```
int *ptr;  
int arr[5];  
  
// store the address of the first  
// element of arr in ptr  
ptr = arr;
```

Here, ptr is a pointer variable while arr is an int array. The code ptr = arr; stores the address of the first element of the array in variable ptr.

Notice that we have used arr instead of &arr[0]. This is because both are the same. So, the code below is the same as the code above.

```
int *ptr;  
int arr[5];  
ptr = &arr[0];
```

The addresses for the rest of the array elements are given by &arr[1], &arr[2], &arr[3], and &arr[4].

## C++ Pointers

### Address in C++

If we have a variable `var` in our program, `&var` will give us its address in the memory. For example,

### Example 1: Printing Variable Addresses in C++

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

int main()
{
    // declare variables
    int var1 = 3;
    int var2 = 24;
    int var3 = 17;

    // print address of var1
    cout << "Address of var1: " << &var1 << endl;

    // print address of var2
    cout << "Address of var2: " << &var2 << endl;

    // print address of var3
    cout << "Address of var3: " << &var3 << endl;
}
```



Here, 0x at the beginning represents the address is in the hexadecimal form.

Notice that the first address differs from the second by 4 bytes and the second address differs from the third by 4 bytes.

This is because the size of an int variable is 4 bytes in a 64-bit system.

## 'this' pointer in C++

### C++ this Pointer

In C++ programming, this is a keyword that refers to the current instance of the class. There can be 3 main usage of this keyword in C++.

It can be used to pass current object as a parameter to another method.

It can be used to refer current class instance variable.

It can be used to declare indexers

## C++ this Pointer Example

Let's see the example of this keyword in C++ that refers to the fields of current class.

```
#include <iostream>
using namespace std;
class Employee {
public:
    int id; //data member (also instance variable)
    string name; //data member(also instance variable)
    float salary;
    Employee(int id, string name, float salary)
    {
        this->id = id;
        this->name = name;
        this->salary = salary;
    }
    void display()
    {
        cout<<id<<" "<<name<<" "<<salary<<endl;
    }
};
int main(void) {
    Employee e1 =Employee(101, "Sonoo", 890000); //creating an object of Employee
    Employee e2=Employee(102, "Nakul", 59000); //creating an object of Employee
    e1.display();
    e2.display();
    return 0;
}
```

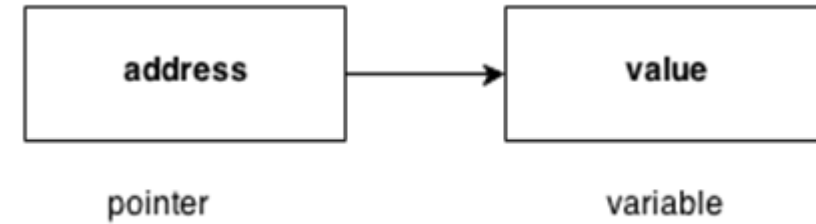
```

#include <iostream.h>
#include<conio.h>
void main()
{
    int a[10][10], b[10][10], mult[10][10], r1, c1, r2, c2, i, j, k;
    cout << "Enter rows and columns for first matrix: ";   cin >> r1 >> c1;
    cout << "Enter rows and columns for second matrix: ";   cin >> r2 >> c2;
    // If column of first matrix is not equal to row of second matrix,
    // ask the user to enter the size of matrix again.
    while (c1!=r2)
    {
        cout << "Error! column of first matrix not equal to row of second.";
        cout << "Enter rows and columns for first matrix: ";cin >> r1 >> c1;
        cout << "Enter rows and columns for second matrix: ";cin >> r2 >> c2;
    }
    // Storing elements of first matrix.
    cout << endl << "Enter elements of matrix 1:" << endl;
    for(i = 0; i < r1; ++i)
        for(j = 0; j < c1; ++j)
        {
            cout << "Enter element a" << i + 1 << j + 1 << " : ";   cin >> a[i][j];
        }
    // Storing elements of second matrix.
    cout << endl << "Enter elements of matrix 2:" << endl;
    for(i = 0; i < r2; ++i)
        for(j = 0; j < c2; ++j)
        {
            cout << "Enter element b" << i + 1 << j + 1 << " : ";   cin >> b[i][j];
        }
    // Initializing elements of matrix mult to 0.
    for(i = 0; i < r1; ++i)
        for(j = 0; j < c2; ++j)
        {
            mult[i][j]=0;
        }
    // Multiplying matrix a and b and storing in array mult.
    for(i = 0; i < r1; ++i)
        for(j = 0; j < c2; ++j)
            for(k = 0; k < c1; ++k)
            {
                mult[i][j] += a[i][k] * b[k][j];
            }
    // Displaying the multiplication of two matrix.
    cout << endl << "Output Matrix: " << endl;
    for(i = 0; i < r1; ++i)
    {
        for(j = 0; j < c2; ++j)
        {
            cout << " " << mult[i][j];

        }   cout << endl;   }   getch();
}

```

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



```
#include <iostream.h>
#include<conio.h>
void main()
{
int number=30;
int * p;
p=&number;//stores the address of number variable
cout<<"Address of number variable is:"<<&number<<endl;
cout<<"Address of p variable is:"<<p<<endl;
cout<<"Value of p variable is:"<<*p<<endl;
    getch();
}
```

### Advantage of pointer

- 1) Pointer reduces the code and improves the performance, it is used to retrieving strings, trees etc. and used with arrays, structures and functions.
- 2) We can return multiple values from function using pointer.

3) It makes you able to access any memory location in the computer's memory.

## Usage of pointer

There are many usage of pointers in C++ language.

### 1) Dynamic memory allocation

In c language, we can dynamically allocate memory using malloc() and calloc() functions where pointer is used.

### 2) Arrays, Functions and Structures

Pointers in c language are widely used in arrays, functions and structures. It reduces the code and improves the performance.

## Pointer Program to swap 2 numbers without using 3rd variable

```
#include <iostream.h>
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;
}
```

```
// C++ Program to display address of each element of an array
```

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

```
int main()
```

```
{
    float arr[3];
```

```
    // declare pointer variable
    float *ptr;
```

```
    cout << "Displaying address using arrays: " << endl;
```

```
    // use for loop to print addresses of all array elements
    for (int i = 0; i < 3; ++i)
    {
        cout << "&arr[" << i << "] = " << &arr[i] << endl;
    }
```

```
    // ptr = &arr[0]
    ptr = arr;
```

```
    cout<<"\nDisplaying address using pointers: "<< endl;
```

```
    // use for loop to print addresses of all array elements
    // using pointer notation
    for (int i = 0; i < 3; ++i)
    {
        cout << "ptr + " << i << " = " << ptr + i << endl;
    }
```

```
    return 0;
```

```
}
```

















