

# C++ Tutorials

(01)

Q ⇒ What is C++? C++ Introduction and History.

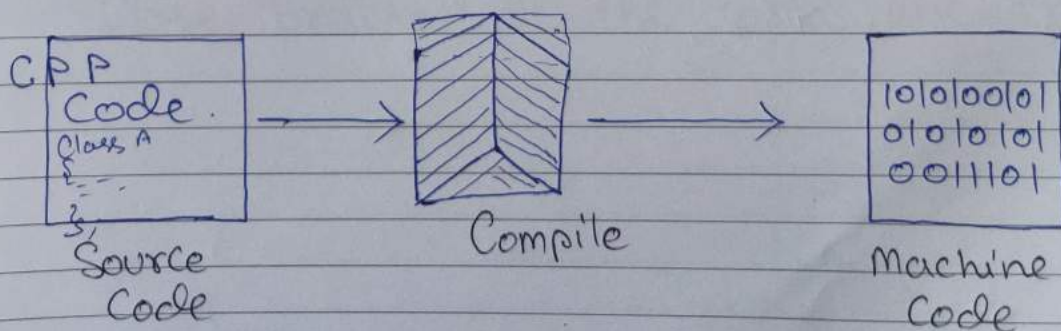
Ans ⇒ C++ is a high-level "Semi-object oriented" programming language developed by "Bjarne Stroustrup" in 1979 at "Bell Labs."

⇒ C++ is a "general purpose", "case-sensitive", "free-form" programming language that supports "object-oriented", procedural and generic programming.

⇒ C++ is a "middle-level" language as it encapsulates both high and low level language features.

⇒ In earlier the name of C++ was "C with Classes".

⇒ It is an imperative and a compiled language.



## C++ Tutorials

(02)

Q ⇒ Define Syntax of C++

#include <iostream.h> } → header file

#include <conio.h>

using namespace std; → Namespace

return-type main() → main method

{ → opening bracket for main function

/\* multi line  
Comment \*/ } → multi comment

Print

Statement { cout << "output statement";

// single line comment } → single comment

return value of return;



main function return  
statement

}



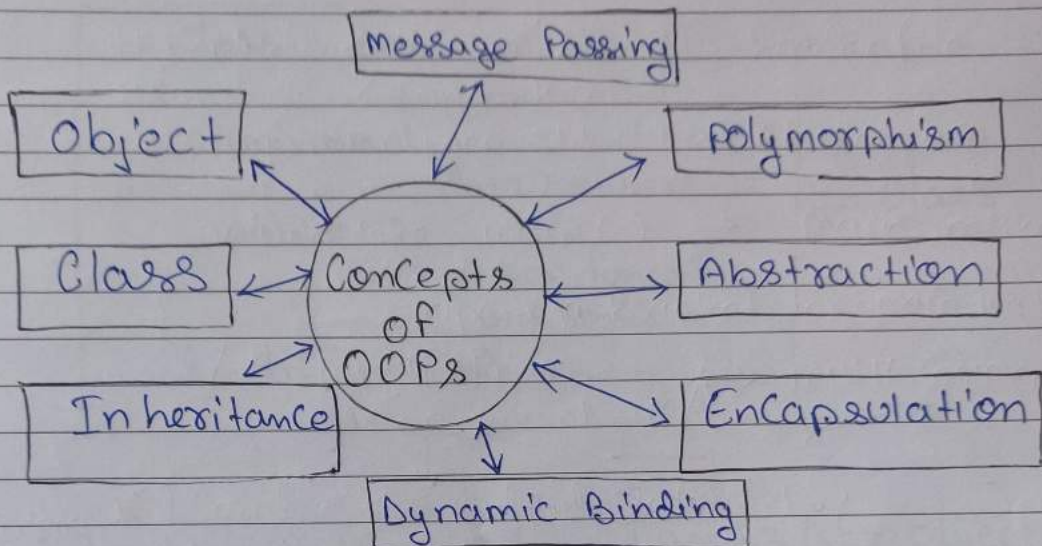
Close bracket of the main function

## C++ Tutorials

(3)

m.I

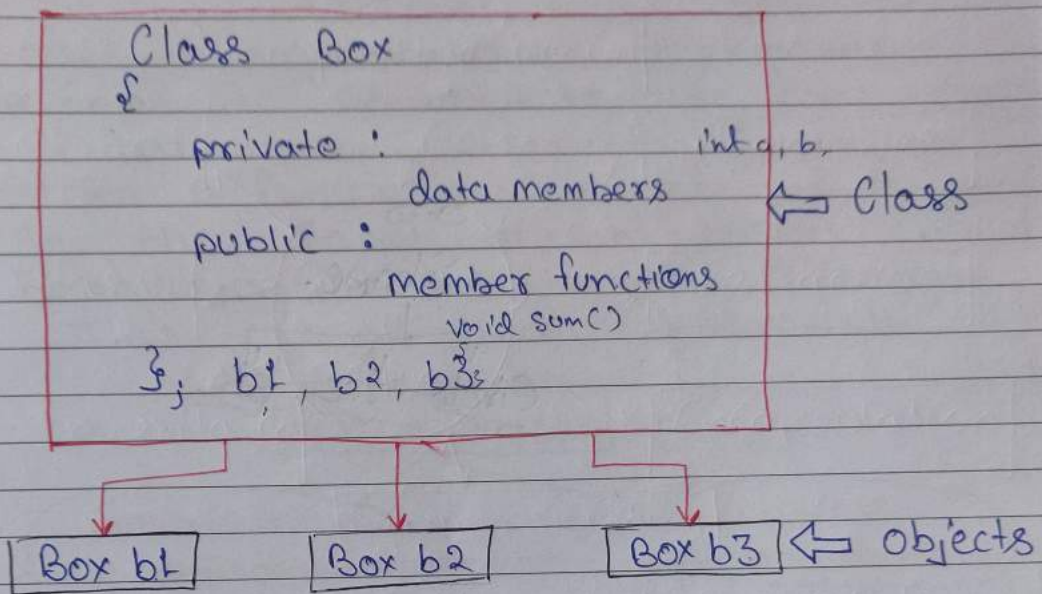
Q ⇒ What is C++ OOPs Concepts? Explain.



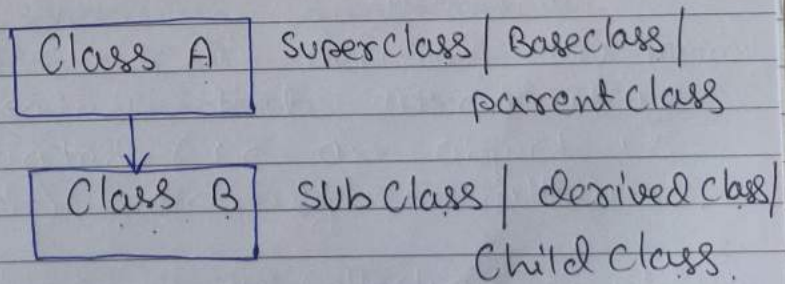
- 1) Object ⇒ Any Entity that has State and behavior is known as Object.
- ◆ Object is a "Active" Entity.
  - ◆ An object is an instance of a class
  - ◆ When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

- 2) Class ⇒ Class is a Collection of Objects.
- ◆ Class is a "Passive" Entity.
  - ◆ A class is a user-defined data type which has data members and member functions.

Ex ⇒ Class and object.



3) Inheritance ⇒ When One Class Access the property (data member and member function) of another class is called Inheritance.



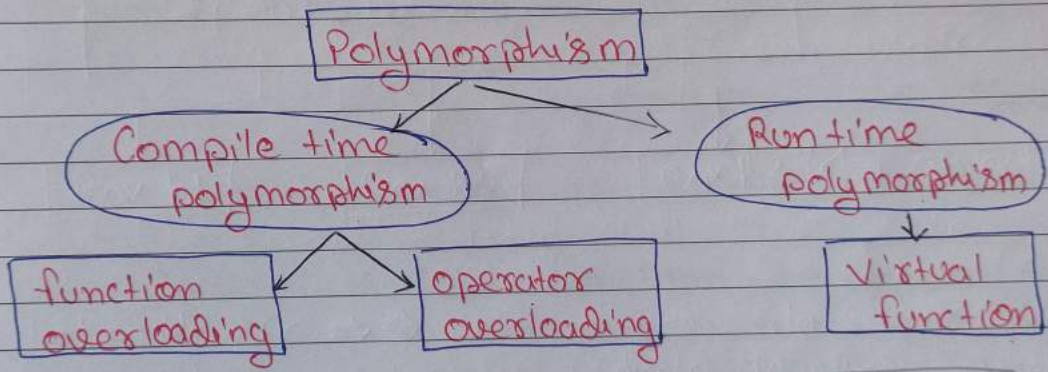
Syntax ⇒

```

Class A
{
  --- data member
  --- mem f-
};

Class B : public A
{
  ---
};
  
```

4) Polymorphism  $\Rightarrow$  When one task is performed as by different ways are known as polymorphism. poly + morphism  
 for ex  $\Rightarrow$  A person at the same time can have different character like a father, a husband and employee. So the same person poses different behaviour in different situations. This is called Polymorphism.



Kailash Joshi

5) Abstraction  $\Rightarrow$  Hiding internal details and showing functionality.  
 for ex  $\Rightarrow$  Phone Call, we don't know the internal processing.

Note  $\Rightarrow$  In C++ we use abstract class and interface to achieve abstraction.

6) Encapsulation  $\Rightarrow$  Binding (or wrapping)

Code and data together into a single unit is known as Encapsulation.

for ex  $\Rightarrow$  Capsule, It is wrapped with different medicines.

7) Dynamic Binding  $\Rightarrow$  In dynamic Binding,

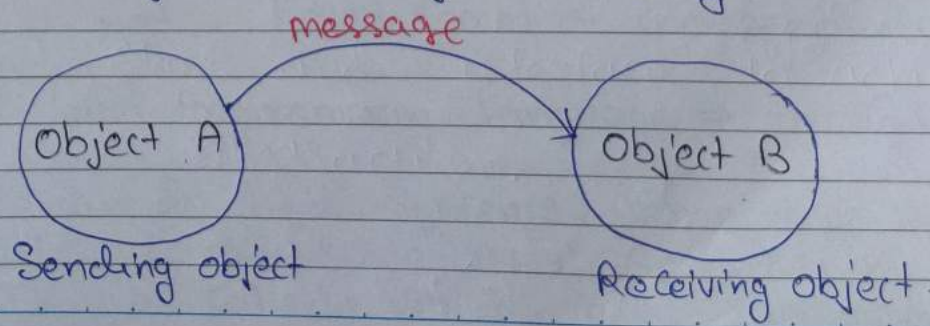
the code to be executed in response to function call is decided at runtime.

$\diamond$  C++ has virtual function to support this.

Kailash Joshi

8) Message Passing  $\Rightarrow$  Objects Communicate

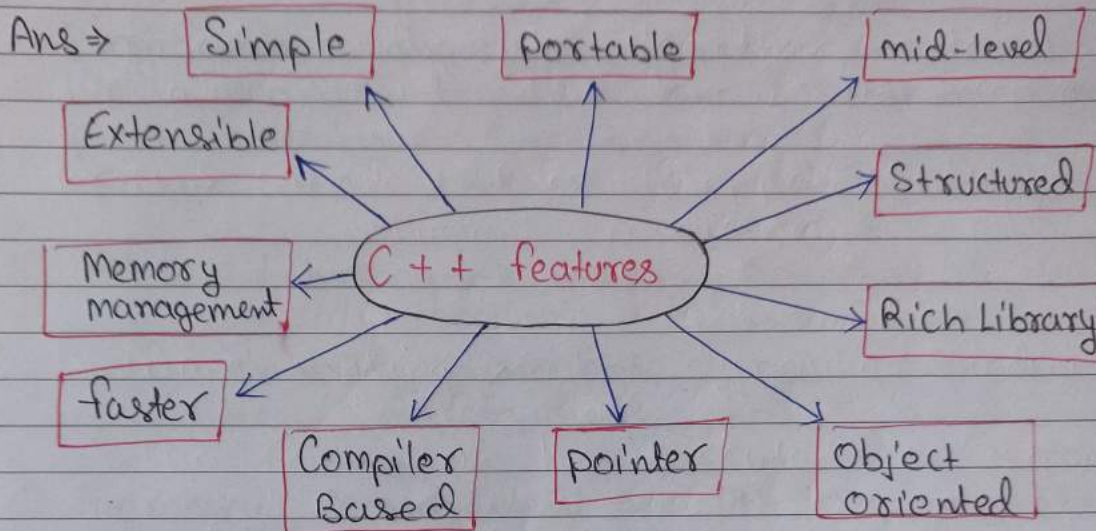
with one another by sending and receiving information to each other through message passing.



## C++ Tutorials

(07)

Q ⇒ Explain Different features of C++.



1) Simple ⇒ It is a simple language in the sense that programs can be broken down into logical unit and parts, has a rich library support. kailash Jasti

2) Machine Independent or Portable ⇒ C Programs can be executed in many machines with little bit or no change. But it is not platform - independent.

3) Mid-level Programming Language ⇒ through C++

We can do both Systems - Programming and build large-scale user Application so It is called mid-level programming.

## C++ Tutorials

(08)

- 4) Structured Programming Language  $\Rightarrow$  C++ is a Structure Programming Language that means we can break the program into parts using functions, so, it is easy to understand and modify.
- 5) Rich Library  $\Rightarrow$  C++ provides a lot of inbuilt functions that makes the development fast.
- 6) Memory Management  $\Rightarrow$  It support the feature of Dynamic memory Allocation. In C++, we can free the allocated memory at any time by calling the free function.
- 7) Speed  $\Rightarrow$  The compilation and execution time of C++ language is fast. Kailash Joshi
- 8) Pointer  $\Rightarrow$  C++ provides the feature of pointer. We can directly interact with the memory by using the pointer.
- 9) Extensible  $\Rightarrow$  It is extensible because it can easily adopt new features.
- 10) object-oriented programming  $\Rightarrow$



## C++ Tutorials

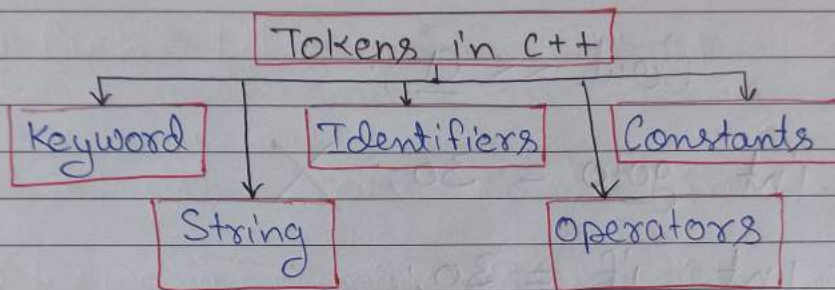
CSE Gyan

09

Q ⇒ What do you mean by C++ Tokens?

Ans ⇒ • A Token is the smallest element of a program that is meaningful to the compiler.

- Tokens act as building blocks of a program.



1) Keywords ⇒ • Keywords are reserved words which have fixed meaning, and its meaning cannot be changed.

- The meaning and working of these keywords are already known to the compiler.

Kailash Joshi

Some keywords are: →

auto, case, double, break, else, if, for, goto, do, long, static, void, return, int, switch, union, while, char etc.

## C++ Tutorials

(10)

2) Identifiers  $\Rightarrow$  Identifiers refer to the name of variables, functions, arrays, classes etc. created by programmer.

Rules to define Identifiers in Each language.

**Rule 1:** Only alphabetic characters, digits and underscores are permitted.

**Rule 2:** The name cannot start with a digit.

**Rule 3:** Uppercase and lowercase letters are distinct.

**Rule 4:** A declared-keyword cannot be used as a variable name.

Kailash Joshi

3) Constants  $\Rightarrow$  Constants are like a variable except that their value never changes during execution once defined.

C++ support several kinds of literal constant. They include integers, characters, floating point number and string.

```
234 // decimal integer
12.34 // floating point integer
037 // Octal integer
0x2 // hexadecimal integer
"C++" // string integer constant
'A' // character constant
```

Camlin

## C++ Tutorials

(11)

- 4) Strings  $\Rightarrow$  • String in C++ are used to store letters and digits.
- String can be referred to as an array of characters as well as individual data type.

Ex  $\Rightarrow$

```
Char name[30] = "Hello";  
or  
String name = "Hello";
```

- 5) Operators  $\Rightarrow$  C++ operator is a symbol that is used to perform mathematical or logical manipulations.

Types of Operators:

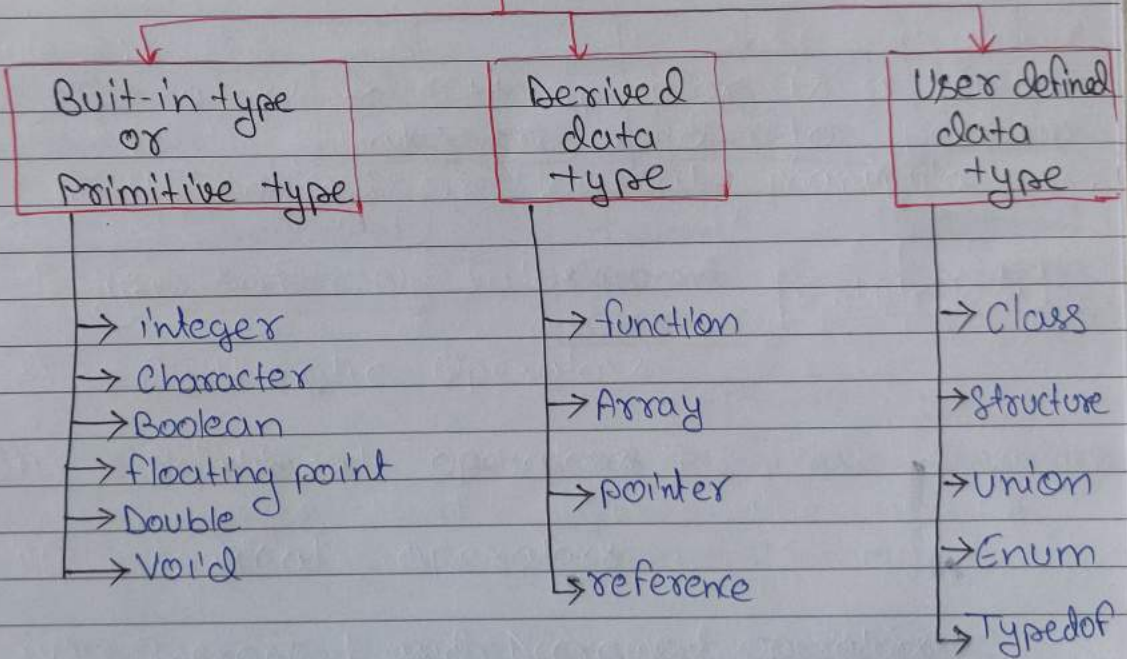
Kaibeh Joshi

- i) Arithmetic Operators
- ii) Relational operators
- iii) Logical operators
- iv) Increment and Decrement operators ++ --
- v) Bitwise operators & |
- vi) Assignment operators =
- vii) Conditional operators ?

Q ⇒ What do you mean by data types in C++?

Ans ⇒ Data types define the type of data a variable can hold.  
There are Three types of data types in C++.

Data types in C++



NOTE ⇒ Use sizeof() function to find all primitive data type size.

Ex ⇒ sizeof(char) ⇒ 1 (in byte)

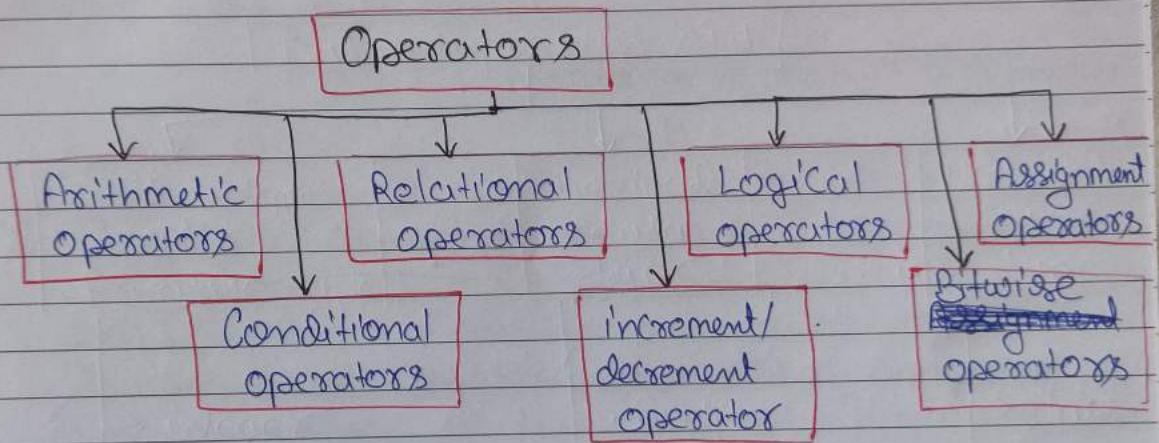
# C++ Tutorials

(13)

Q) What do you mean by Operators in C++?

Sol<sup>n</sup> ⇒ An Operator is a Symbol that tells the Compiler to perform some specific mathematical or logical manipulations.

C++ Provide the following Types of Operators-



Kailash Jishi

1) Arithmetic Operators ⇒ These Operators

are used to perform arithmetic/mathematical operations on operands

Ex ⇒ +, -, \*, /, %

$$\begin{array}{r} 6 \\ 2 \overline{) 13} \\ \underline{12} \\ 1 \end{array}$$

↓ Qu      ↗ Remainder

$$13 / 2 = 6$$
$$13 \% 2 = 1$$

## C++ Tutorials

(14)

2) Relational operators  $\Rightarrow$  These are used for  
The Comparison of the values of two operands

Ex  $\Rightarrow$   $==$ ,  $>=$ ,  $<=$ ,  $>$ ,  $<$

```
int a = 3;  
int b = 5;
```

$a < b$ ; // operator to check if a is smaller than b.

3) Logical operators  $\Rightarrow$  Logical Operators

are used to Combining two or more  
Conditions / Constraints or to Complement  
the Evaluation of the original Condition in  
Consideration.

Kailash Joshi

The result of the operation of a logical  
operator is a Boolean value either True  
or false.

There are three logical operator.

1)  $\&\&$  // AND operator

2)  $\|\|$  // OR operator

3)  $!$  // NOT operator

$a > b \&\& a > c$

$a > b \|\| a > c$

Kailash Joshi

Camlin

## C++ Tutorials

(15)

4) Assignment Operators:  $\Rightarrow$  These operators

are used to assigning value to a variable. The left side operand of the assignment operator is a variable and the right side operand of the assignment operator is a value.

Ex  $\Rightarrow$   $a = 10;$

~~$a = b$~~

$a += b; \Rightarrow a = a + b$

5) Conditional operators:  $\Rightarrow$  The Conditional operator is kind of

similar to the if-else statement as it does follow the same algorithm as of if-else statement, but the Conditional operator takes less space and helps to write the if-else statements in the shortest way possible.

It is also known as Ternary operator.

Symbol ( ? : )

Keikeyeh Jishi

Syntax  $\Rightarrow$

Variable = Expression ? true statement : false statement ;

Ex  $\Rightarrow$

$\text{int } a = (b > c) ? b : c ;$

## C++ Tutorials

(16)

6) Increment/Decrement Operator  $\Rightarrow$  increment Operator ( $++$ ) is used to increase the value of the operand by 1. Whereas the decrement Operator ( $--$ ) is used to decrease the value of operand by 1.

Ex  $\Rightarrow$   $x++$ ; // Same as  $x = x + 1$

$x--$ ; // Same as  $x = x - 1$

Kailash Joshi

7) Bitwise Operators  $\Rightarrow$  The Bitwise operators

are used to perform bit level operations on the operands. The operators are first converted to bit-level and then the calculation is performed on the operands.

Some Bitwise operators are :-

1)  $\&$  // Bitwise AND operator

2)  $|$  // Bitwise OR operator

3)  $\wedge$  // Bitwise XOR operator

4)  $\sim$  // Bitwise Ones Complement operator.



v.v.I

Q. What do you mean by Type Casting / Type Conversion in C++?

Ans ⇒ A Type Casting or Type Conversion basically a Conversion from one Primitive data type to another primitive data type.

# There are two types of type Casting

1) Implicit Type Casting / automatic type Casting.

Kailash Joshi

2) Explicit type Casting / manual type Casting.

1) Implicit type Casting ⇒ In implicit or automatic Casting Compiler will automatically change one type of data into another.

Char → short int → int → unsigned int → long int →  
float → double → long double etc.

## C++ Tutorials

18

```
Ex ⇒ #include <iostream>
       using namespace std;
       int main ()
       {
           short x = 20;
           int y = x; // implicit Type Casting
           cout << "the value of x" << x << endl;
           cout << "The value of y" << y;
           return 0;
       }
```

2) Explicit Type Casting ⇒ When the user manually changes data from one type to another, This is known as Explicit type Casting.

Kailash Joshi

Syntax ⇒ (data type) Expression;

long double → double → float → long int →  
unsigned int → int → short int → char

## C++ Tutorials

19

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

      int main()
      {
        double x = 1.2;

        int y = (int) x; // Explicit type Casting

        cout << "value of y = " << y;

        return 0;
      }
```

# C++ Tutorials

20

CS Engineering gyan

## C++ Classes and Objects:

Class ⇒ A Class in C++ is the building blocks that leads to object-oriented programming.

Kailash Joshi

- Class is a user-defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class.
- A C++ class is like a blueprint for an object.
- A Class is a user defined data type which has data members and member functions.

Syntax: ⇒

```
Keyword — Class ClassName — user-defined Name
{
  Access specifier: // Can be private, public or Protected
  Data Members; // Variables to be used
  Member functions() // methods to access data member
  {
    Body of function;
  }
}; // class name end with a semicolon
```

## C++ Tutorials

(21)

cs Engineering gpn

Object ⇒ An object is an instance of a class. When a class is defined, no memory is allocated but when it is instantiated (such as an object is created) memory is allocated. Kailash Joshi

Declaring object ⇒ When a class is defined, only the specification for the object is defined, no memory or storage is allocated.

To use the data and access functions defined in the class, you need to create object.

Syntax ⇒

classname objectName;

Access data members and member functions ⇒

The data members and member function of class can be accessed using dot (.) operator with object.

Syntax ⇒

objectName.memberdata;

objectName.memberfunction();

## C++ Tutorials

(22)

CS Engineering gyan

\*WAP for class and object and access data member and member function.

```
# include <iostream>
```

```
using NameSpace std;
```

```
class Simple
```

```
{
```

```
public:
```

```
String name;
```

```
void display();
```

```
{
```

```
cout << "your name is:" << name;
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
Simple ob;
```

```
ob.name = "kailash";
```

```
ob.display();
```

```
return 0;
```

```
}
```

Kailash Jorhi

## C++ Tutorials

23

CS Engineering gyan

# WAP in c++ for more than class and object.

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

```
Class Simple1
```

```
{
```

```
public:
```

```
void display()
```

```
{
```

```
cout << "display function call";
```

```
}
```

```
};
```

```
Class Simple2
```

```
{
```

```
public:
```

```
void show()
```

```
{
```

```
cout << "show function call";
```

```
}
```

```
};
```

```
int main()
```

```
{
```

```
Simple1 ob1;
```

```
ob1.display();
```

```
cout << "\n";
```

```
Simple2 ob2;
```

```
ob2.display show();
```

```
return 0;
```

```
}
```

Kailash Joshi

# C++ Tutorials

24

CS Engineering gyan

## Access Specifiers in C++

- Access specifiers define how the members (attributes and methods) of a class can be accessed.
- Access modifiers are used to implement an important aspect of object-oriented programming known as **Data Hiding**.
- There are three access modifier in C++

1) **Public**: members are accessible from outside the class.

2) **Private**: members cannot be accessed or view from outside the class.

3) **Protected**: members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

Syntax ⇒

Class Classname

{

public:

Data member;  
member function;

private:

Data member;  
member function;

protected:

Data member;  
member function;

};

Kaishav Joshi



## C++ Tutorials

25

CS Engineering gyan

### Class Methods in C++ ⇒

- Method are functions that belongs to the class.
- There are two ways to define functions that belongs to a class.

1) Inside class definition / inline

2) outside class definition

1) Inside class definition ⇒

- The member function is defined inside the class definition it can be defined directly.
- Similar to accessing a data member in the class, we can also access the public member functions through the class object using the **dot operator (.)**.

Syntax ⇒

```
class className {
    public:
    returnType methodName() // method
    {                          // inside
        Body of member function // class
    }                          // definition
};
```

Kailash Joshi

2) Outside Class definition  $\Rightarrow$  (methods)  $\Rightarrow$

To define a function outside the class definition, you have to declare it inside the class then define it outside of the class. This is done by specifying the name of the class, followed by the **scope resolution :: operator**, followed by the name of the function.

**NOTE**  $\Rightarrow$  You access methods just like you access attributes, by creating an object of the class and using the **dot syntax (.)**.

Syntax  $\Rightarrow$

```
class classname()
```

```
{
```

```
public:
```

```
returntype methodName();
```

```
}
```

// method / function definition outside the class

```
returntype classname :: methodName()
```

```
{
```

```
Body of the method;
```

```
}
```

// Then create object of class and call method.

Constructors in C++

- A Constructor is a special type of member function of a class which initializes object of a class.
  - In C++, Constructor is automatically called when object is created.
  - It is special member function of a class because it does not have any return type.
- # How Constructors are different from normal member function?

Kaishik Joshi

- 1) Constructor has same name as the class itself.
- 2) Constructor don't have return type.
- 3) A Constructor is automatically called when an object is created.
- 4) Default Constructor don't have input argument however, Copy and Parameterized Constructors have input arguments.
- 5) Constructor must be placed in public section of class.
- 6) Constructor used to initialize the data members of new object generally.

## C++ Tutorials

(28)

# WAP to Create a Constructor in C++

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

```
class ConstructorTest. // The class
{
public: // access specifier
```

```
    ConstructorTest() // Constructor
    {
        cout << "Constructor Created",;
    }
}
```

```
};
```

Kailash Joshi

```
int main()
```

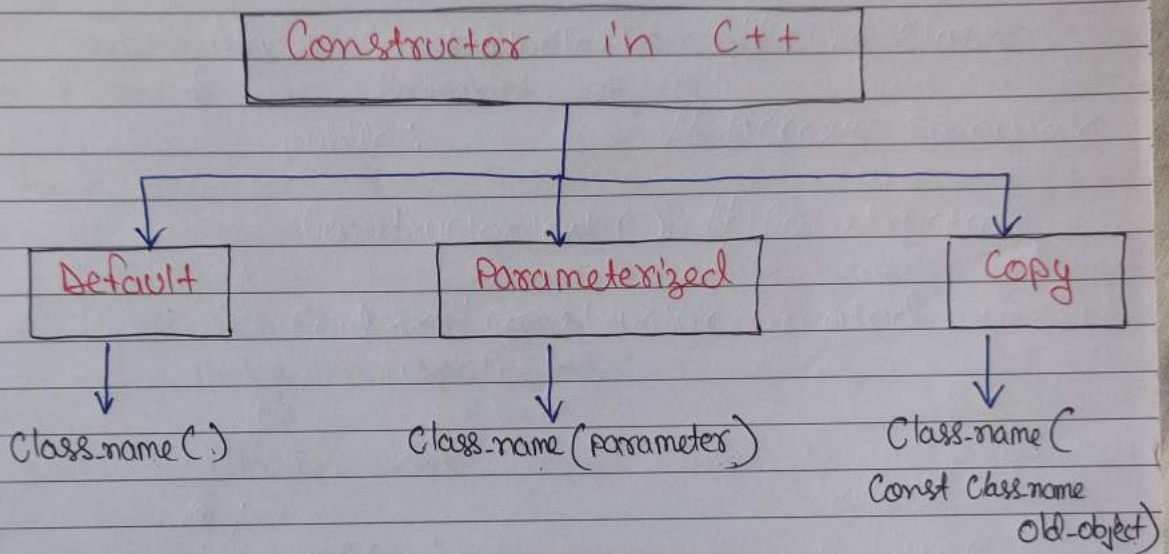
```
{
```

```
    ConstructorTest ob; // Create an
                        // object of ConstructorTest
                        // (this will call the constructor)
    return 0;
```

```
}
```

Types of Constructor in C++

There are three types of constructor in C++.



# default Constructor ⇒ Default Constructor is the constructor which does not take any argument, it has no parameters.

Syntax ⇒

```
Class Classname
{
    public:
    Classname()
    {
        member data,
        member function,
    }
};
```

Kailesh Joshi

## C++ Tutorials

30

### C++ Parameterized Constructor

- A Constructor which has parameters is called parameterized constructor.
- Constructor is used to provide different value to distinct object.

Ex ⇒ WAP to add Two no. Using Parameterized Constructor.

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

Kaifush Joshi

```
class add
```

```
{
```

```
public:
```

```
add (int a, int b)
```

```
// Constructor  
with Parameter
```

```
{
```

```
int c = a + b;
```

```
cout << "sum = " << c;
```

```
}
```

```
};
```

```
int main ()
```

```
{
```

```
add ob (10, 20);
```

```
// constructor  
initiated
```

```
return 0;
```

```
}
```

## C++ Tutorials

(31)

### C++ Copy Constructor

- A Copy Constructor is an Overloaded Constructor used to declare and initialize an object from another object.

There are two types of Copy Constructor

1) Default Copy Constructor  $\Rightarrow$  The Compiler defines the default Copy Constructor. If the user defines no Copy Constructor, Compiler supplies its Constructor.

2) User defined Constructor  $\Rightarrow$  The programmer defines the user-defined Constructor

Syntax of User-define Copy Constructor  $\Rightarrow$

Class name (const Class-name &old-object);

Ex  $\Rightarrow$

Class A

Kailash Jishi

{

A(A &x) // Copy Constructor

{

// Copy Constructor body

}

};

Copy Constructor can be called in following ways:

A a2(a1)

A a2 = a1

} a1 initializes the a2 object.

## C++ Tutorials

32

# WAP in C++ for Copy Constructor.

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

```
class CopyTest
{
```

```
public:
```

```
int x;
```

```
CopyTest (int a) // parameterized  
                constructor
```

```
{
```

```
x = a;
```

```
}
```

```
CopyTest (CopyTest &i) // Copy constructor
```

```
{
```

```
x = i.x;
```

```
}
```

```
};
```

Kailash Joshi

```
int main()
```

```
{
```

```
CopyTest a1(20); // calling the parameterized  
                constructor
```

```
CopyTest a2(a1); // calling the Copy constructor
```

```
cout << a2.x;
```

```
return 0;
```

```
}
```



Destructor in C++

- A destructor work opposite to Constructor. It destructs the objects of classes.
- A destructor is defined like Constructor. It must have same name as class. But it is prefixed with a **tilde sign (~)**
- Destructor function is automatically invoked when the objects are destroyed.
- It cannot be declared static or Const.
- The destructor does not have arguments.
- It has no return type not even void.
- destructor should be declared in the public section of the class.
- The programmer cannot access the address of destructor.

Syntax ⇒

Kailash Joshi

```
class classname
{
    public:
    classname() // constructor
    { }
    ~classname() // destructor
    { }
};
```

## C++ Tutorials

34

# WAP for destructor in C++;

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

```
class DestructorTest
```

```
{
```

```
public:
```

```
DestructorTest () // constructor
```

```
{
```

```
cout << "Constructor Invoked" << "\n";
```

```
}
```

```
~DestructorTest () // destructor
```

```
{
```

```
cout << "Destructor Invoked";
```

```
}
```

```
};
```

Kailash Jishi

```
int main()
```

```
{
```

```
DestructorTest ob;
```

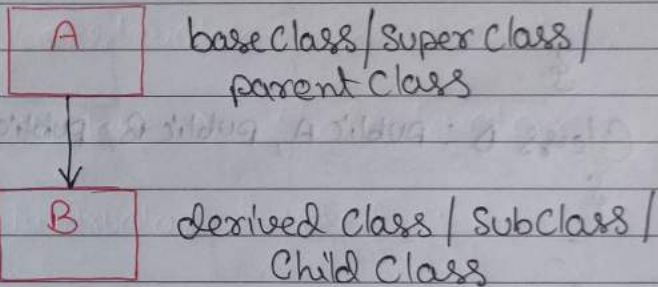
```
return 0;
```

```
}
```

Inheritance in C++

The Capability of a class to derive properties and characteristics from another class is called inheritance.

Inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically.



the class which inherits the members of another class is called **derived class** and the class whose members are inherited is called **base class**.

Syntax ⇒

Itailash Joshi

```
class derived-class-name : visibility-mode base-class-name
```

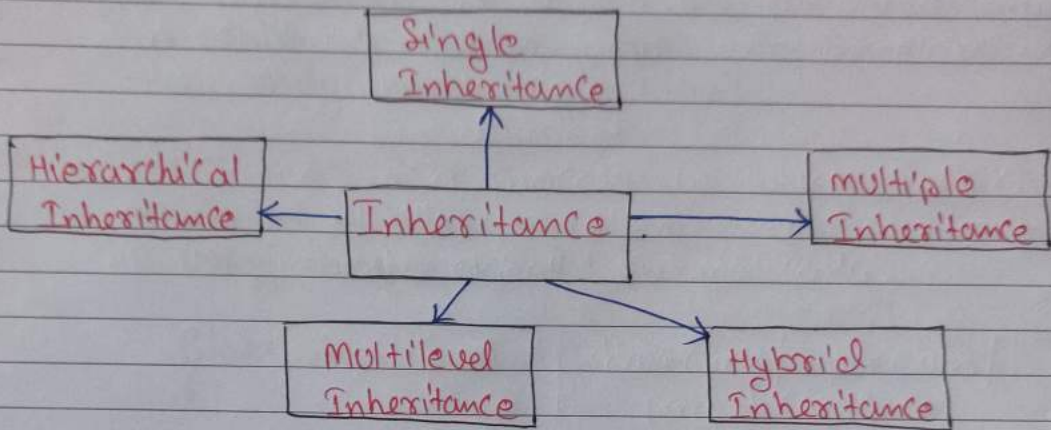
```
{
```

```
// body of the derived class.
```

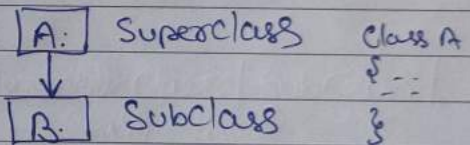
```
}
```

**visibility-mode** ⇒ The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

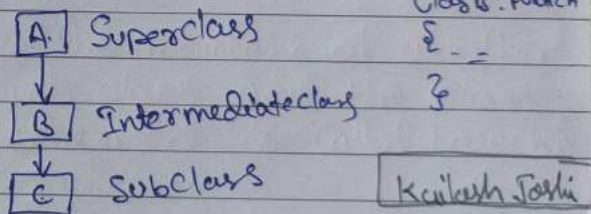
Types of Inheritance



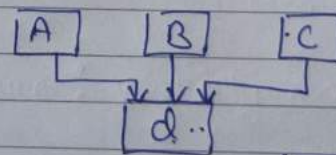
1) Single Inheritance



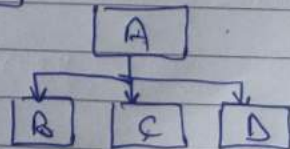
2) multilevel Inheritance



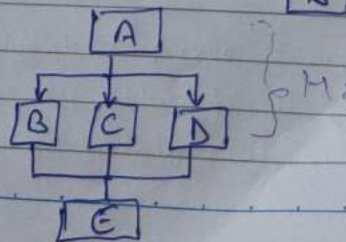
3) multiple Inheritance



4) Hierarchical Inheritance



5) Hybrid inheritance



C++ Single Inheritance

- Single inheritance is defined as the inheritance in which a derived class is inherited from the only one base class.

## # WAP for Single Inheritance

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

```
Class A
```

```
{
```

```
public;
```

```
void display ()
```

```
{
```

```
cout << "super class display function \n";
```

```
}
```

```
};
```

```
Class B : public A
```

```
{
```

```
public;
```

```
void show ()
```

```
{
```

```
cout << "subclass show function";
```

```
}
```

```
};
```

```
int main ()
```

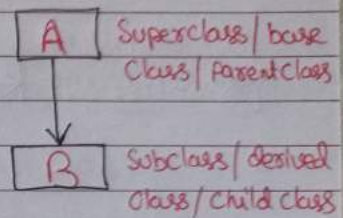
```
{
```

```
B ob;
```

```
ob.display();
```

```
ob.show();
```

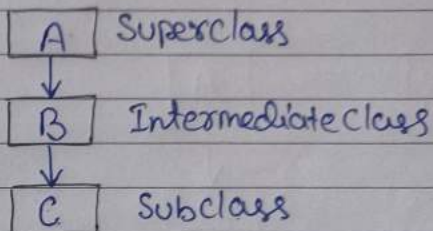
```
}
```



Kailesh Joshi

C++ Multi Level Inheritance

- Multilevel Inheritance is a process of deriving a class from another derived class.
- When one class inherits another class which is further inherited by another class, it is known as multi-level inheritance in C++.



## # WAP for Multi-level Inheritance

# include &lt;iostream&gt;

using namespace std;

class A

{

public:

void display()

{

cout &lt;&lt; "super class display function\n";

}

};

class B: public A

{

public:

void show()

{

cout &lt;&lt; "Intermediate show function\n";

}

};

Kailash Jishi

## C++ Tutorials

39

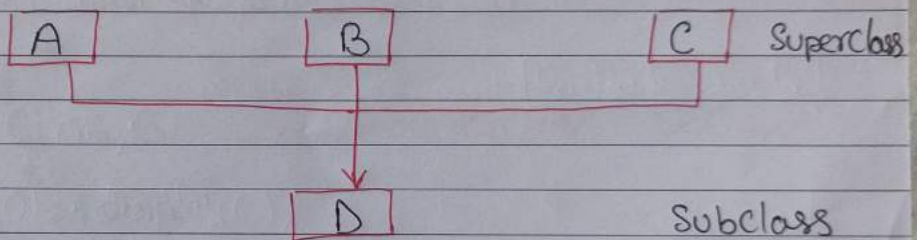
```
Class C : public B
{
    public:
    void display1()
    {
        cout << "Subclass display1 function";
    }
};
```

```
int main()
{
    C ob;
    ob.display();
    ob.show();
    ob.display1();
    return 0;
}
```

Kailash Jishi

C++ multiple Inheritance

- multiple inheritance is the process of deriving a new class that inherits the attributes from two or more classes.
- One sub class Access the properties of more than one Superclass are called multiple Inheritance.



Syntax ⇒

```
Class subclass : visibility Superclass 1, visibility  
                Superclass 2 ...  
{  
    // Body of the class;  
}
```

Kailesh Joshi



## C++ Tutorials

(41)

# WAP for multiple Inheritance in C++

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

```
class A // superclass first
{
public:
void display()
{
cout << "Super class A \n";
}
};
```

```
class B // superclass second
{
public:
void show()
{
cout << "Super class B \n";
}
};
```

```
class C : public A, public B // subclass C
// inherit A and B
// superclass properties
{
public:
void display1()
{
cout << "Subclass C";
}
};
```

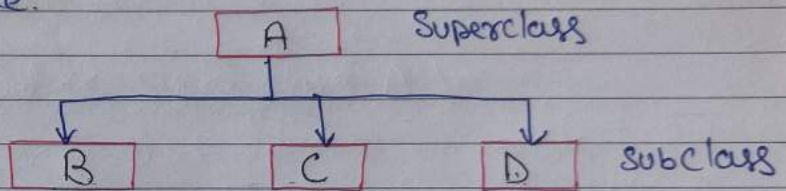
```
int main()
{
C ob;
ob.display();
}
```

```
ob.show();
ob.display1();
return 0;
```

```
}
```

< C++ Hierarchical Inheritance

- In Hierarchical Inheritance, more than one sub class is inherited from a single base class.
- When one superclass property access more than one subclass is called Hierarchical Inheritance.



Syntax =>

Class A

```
{  
  == statement;  
};
```

Kailash Joshi

Class B : access-specifier A // derived class from A

```
{  
  == statement;  
};
```

Class C : access-specifier A // derived class from A

```
{  
  == statement;  
};
```

Class D : access-specifier A // derived class from A

```
{  
  == statement;  
};
```

## C++ Tutorials

(43)

// WAP for Hierarchical Inheritance in C++

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

```
class A // superclass A
{
public:
void display()
{
cout << "super class display function \n";
}
};
```

```
class B: public A // subclass B
{
public:
void show()
{
cout << "subclass show method \n";
}
};
```

```
class C: public A // subclass C
{
public:
void display1()
{
cout << "subclass display 1 method \n";
}
};
```

Kaishash Joshi

## C++ Tutorials

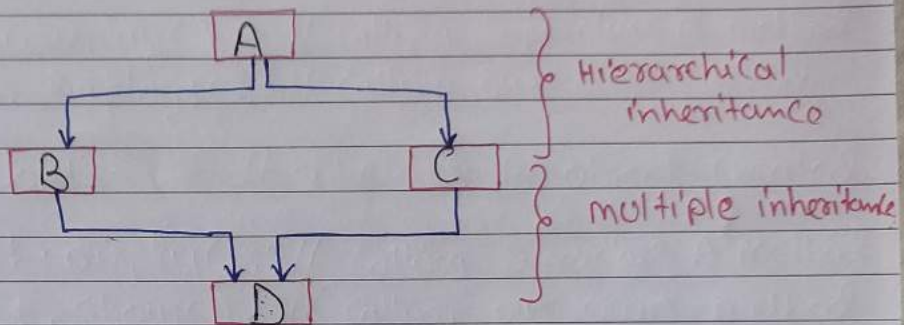
44

```
int main()
{
    C ob; // subclass C object created
    ob.display(); // calling superclass A method
    ob.displayt(); // calling our own method
    B obt; // subclass B object created
    obt.display(); // calling superclass A method
    obt.show(); // calling our own method
    return 0;
}
```

Kailash Joshi

C++ Hybrid Inheritance

- Hybrid inheritance is a combination of more than one type of Inheritance.



Syntax ⇒

```
Class A  
{  
    Statement;  
};
```

```
Class B : public A  
{  
    Statement;  
};
```

```
Class C : public A  
{  
    Statement;  
};
```

```
Class D : public B, public C  
{  
    Statement;  
};
```

Kailash Joshi

## C++ Tutorials

(46)

# WAP for Hybrid Inheritance in C++

```
#include <iostream>
```

```
using namespace std;
```

```
class A // Superclass
```

```
{
```

```
public:
```

```
void display()
```

```
{
```

```
cout << "Super class display function \n",
```

```
}
```

```
};
```

```
class B: public A // subclass
```

```
{
```

```
public:
```

```
void show()
```

```
{
```

```
cout << "subclass show function \n",
```

```
}
```

```
};
```

```
class C: public A // subclass
```

```
{
```

```
public:
```

```
void display1()
```

```
{
```

```
cout << "subclass display 1 function \n",
```

```
}
```

```
};
```

Kailash Joshi

## C++ Tutorials

(47)

```
Class D: public B, public C // Sub-Sub Class
{
    public:
    void display2()
    {
        cout << "sub-subclass display2 function\n";
    }
};
```

```
int main()
```

Kailash Joshi

```
{
```

```
    D ob; // Sub-sub Class object created
```

```
    ob.display1();
```

```
    ob.show();
```

```
    ob.display2();
```

```
    B ob1; // Sub Class object created
```

```
    ob1.display();
```

```
    return 0;
```

```
}
```

## C++ Tutorials

(48)

### 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 indexes.

### C++ this pointer Example ⇒

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

```
class Employee {
public:
    int id;
    string name;
    float salary;
```

```
Employee (int id, string name, float salary)
{
    this -> id = id;
    this -> name = name;
    this -> salary = salary;
}
```



```
void display()  
{  
    cout << id << " " << name << " " << salary;  
}  
};
```

```
int main()  
{  
    Employee e1 = Employee ( 100, "Kailash", 20000);  
    e1.display();  
    return 0;  
}
```

## C++ Tutorials

(50)

### C++ Static keyword

- In C++ Static is a keyword or modifier that belongs to the type not instance. So instance is not required to access the static members.
- In C++, static can be field, method, constructor, class, properties, operator and event.
- A field which is declared as static is called static field. Unlike instance field which gets memory each time whenever you create object, there is only one copy of static field created in the memory. It is shared to all the objects.

C++ static field Example ⇒

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

class Account
{
public:
    int accno; // data member (also Instance Variable)
    string name; // data member (Instance Variable)
    static float rateOfInterest; // static variable
    // (not Instance Variable)
    Account (int accno, string name)
    {
    }
};
```

this -> accno = accno;  
this -> name = name;

}  
void display ()

{  
cout << accno;  
cout << name;  
cout << rateOfInterest;

}  
};

float Account :: rateOfInterest = 7.6;

int main ()

{  
Account ob = Account (200, "kailash");  
// Create an object of  
Employee  
ob.display ();  
return 0;  
}

# C++ Tutorials

52

## C++ Structs

- In C++, Classes and Structs are blueprints that are used to create the instance of a class.
- Unlike Class, Structs in C++ are value type than reference type. It's useful if you have data that is not intended to be modified after creation of struct.
- C++ structure is a collection of different data types. It's similar to the class that holds different types of data.

Syntax ⇒

```
struct structure_name  
{  
    // method declarations  
};
```

C++ Struct Example ⇒

```
#include <iostream>  
using namespace std;  
  
struct Rectangle  
{  
    int width, height;  
};
```

```
int main()
{
    struct Rectangle rec;
    rec.width = 8;
    rec.height = 5;
    cout << "Area of rectangle is:" << (rec.width *
        rec.height);
    return 0;
}
```

## C++ Enumeration, Enum

- In C++ programming, enum or enumeration is a data-type consisting of named values like elements, members, etc. that represent integral constants.
- It provides a way to define and group integral constants. It also makes the code easy to maintain and less complex.

### Definition and Declaration of Enum

- To define enum in C++, you must use the enum keyword along with the elements separated by commas. The basic syntax →

```
enum name_of_enum  
{  
    element1,  
    element2,  
    element3,  
    ⋮  
    elementn;  
};
```

Enum Example →

```
#include <iostream>
```

```
using namespace std;
```

```
enum week {
```

```
    Monday, Tuesday, Wednesday,  
    Thursday, Friday, Saturday,  
    Sunday };
```

```
int main()
```

```
{
```

```
    week day;
```

```
    day = Wednesday;
```

```
    cout << "Day: " << day;
```

```
    return 0;
```

```
}
```

Output: Day: 02

M.M.IC++ Friend function

- If a function is defined as a friend function in C++, then the protected and private data of a class can be accessed using the function.
- By using the keyword friend compiler knows the given function is a friend function.

Declaration of friend function in C++

```
class class_name  
{  
    friend data_type function_name (arguments);  
};
```

Characteristics of a friend function:

- The function is not in the scope of the class to which it has been declared as a friend.
- It cannot be called using the object as it is not in the scope of that class.
- It can be invoked like a normal function without using the object.
- It cannot access the member name directly and has to use an object name and dot membership operator with the member name.
- It can be declared either in private or the public part.



### C++ friend function Example

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

Class Box

```
{
  private:
    int length;
  public:
    Box(): length(0) { }
  friend int printlength(Box); // friend function
};
```

```
int printlength (Box b)
{
  b.length += 10;
  return b.length;
}
```

```
int main()
{
  Box b;
  cout << "length of box: " << printlength(b);
  return 0;
}
```

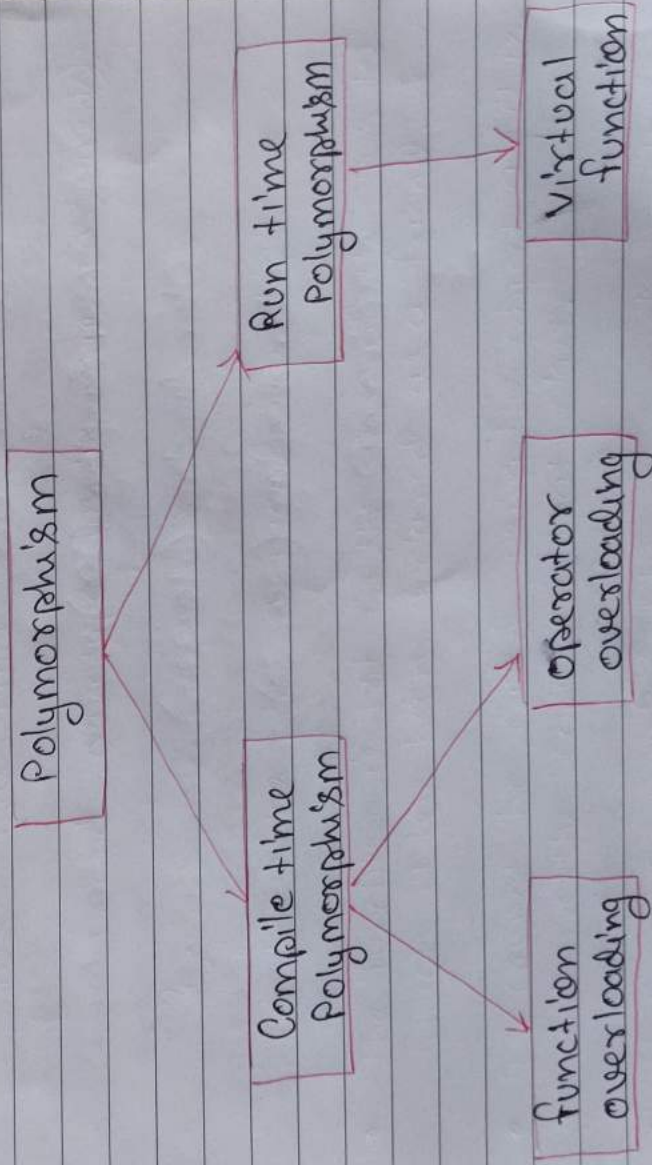
# C++ Tutorials

(58)

~~M.M.S~~

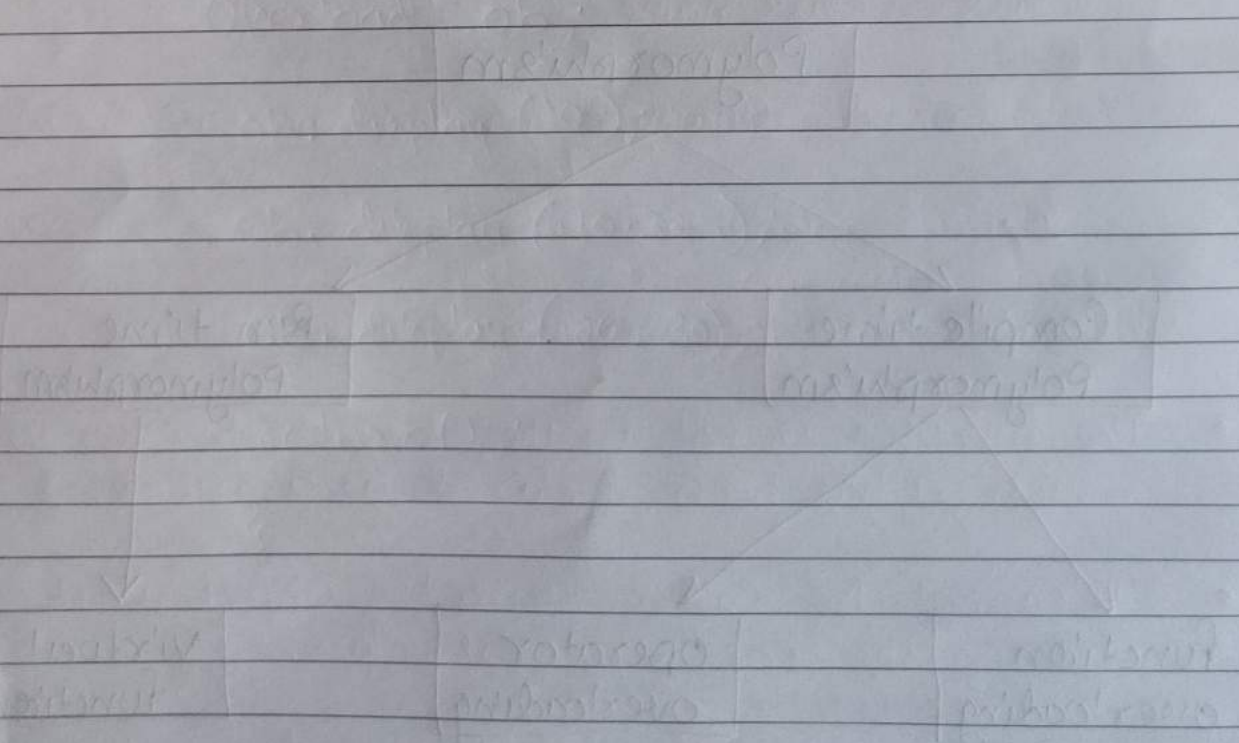
## C++ Polymorphism

- Polymorphism means "many forms" and it occurs when we have many classes that are related to each other by inheritance.
- Polymorphism is the combination of Poly + morph which means many forms.
- It is a greek word that means many forms.



- We can define polymorphism as the ability of the message to be displayed in more than one form.

A real-life example of polymorphism, a person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person poses different behavior in different situations. This is called polymorphism.



# C++ Tutorials

(60)

## Compile time Polymorphism

This type of Polymorphism is achieved by function overloading or operator overloading.

- # function overloading  $\Rightarrow$  • When there are multiple function with same name but different parameters then these functions are said to be overloaded.
- function can be overloaded by change in number of arguments or/and change in type of arguments.

### function overloading Example $\Rightarrow$

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

class overload
{
public:
void display(int x)
{
cout << "value of x = " << x;
}
void display(double x)
{
cout << "value of x = " << x;
}
}
```

```

void display (int x, int y)
{
  cout << "value of x and y is" << x << ", "
    << y;
}
}

```

```

int main()

```

```

{

```

```

  Overload ob;

```

```

  ob.display (10);

```

```

  ob.display (10.325);

```

```

  ob.display (20, 30);

```

```

  return 0;

```

```

}

```

Operator overloading

- Operator overloading is a compile-time polymorphism in which the operator is overloaded to provide the special meaning to the user-defined datatype.
- Operator overloading is used to overload or redefines most of the operators available in C++.
- Operator overloading is used to perform the operation on the user-defined datatype.
- The advantages of operator overloading is to perform different operation on the same operand.

C++ Operator that cannot be overloaded.

- ⇒ Scope resolution operator (::)
- ⇒ sizeof operator
- ⇒ member selection (.)
- ⇒ member pointer operator (\*)
- ⇒ ternary operator (? :)

## Syntax of Operator Overloading

```

return-type class_name :: operator op (argumed-list)
{
    // body of function
}

```

return-type is the type of value return by function.

Class-name is the name of the Class

operator op is an operator function where op is the operator being overloaded, and the operator is the

Keyword:

## C++ Tutorials

(64)

// CPP Program for operator Overloading  
// WAP to add two Complex no.

#include <iostream>  
using namespace std;

class Complex

{

private:

int real, imag;

public:

Complex (int r=0, int i=0)

{

real = r;

imag = i;

}

Complex operator + (Complex const &obj)

{

Complex res;

res.real = real + obj.real;

res.imag = imag + obj.imag;

return res;

}

void print()

{



(65)

cout << real << " + s" << imag;

}

int main()

{

Complex C1(10,5), C2(2,4);

Complex C3 = C1 + C2;

C3.print();

}

Output = 12 + i9

~~10+10~~ 10+5i  
~~2+4~~ 2+4i

66

a+b

## Unary operator overloading

Unary operators operate on single operand or data.

In unary operator function, no arguments should be passed. It works only with one class objects.

Example of unary operators!

- Unary minus (-) operator
- Logical not (!) operator
- Increment (++) and Decrement (--) operator

Example  $\Rightarrow$  how to unary minus operator  
is overloaded.

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

```
class Space
```

```
{
    int x, y;
```

```
public:
```

```
    void getData (int a, int b);
```

```
    void display ();
```

```
    void operator - (); // overload unary
```

```
};
```

mines

(67)

Void space :: getData (int a, int b)

{

x = a,

y = b,

}

Void space :: display ()

{

cout << x,

cout << y,

~~cout << x,~~

}

Void space :: operator - ()

{

x = -x,

y = -y,

~~cout << x,~~

}

int main ()

{

space s,

s.getData (10, -20),

cout << s.display (),

-s, // Activates operator - () function

cout << s.display (),

return 0,

}

10  
-20  
-10  
20



Complex Complex :: operator+(Complex C)

{

Complex temp,

temp.x = x + C.x

temp.y = y + C.y

return (temp);

}

void Complex :: display()

{

cout << x << "+j" << y;

}

int main()

{

Complex C1, C2, C3;

C1 = Complex(2.5, 3.5);

C2 = Complex(1.6, 2.7);

C3 = C1 + C2;

cout << "C1 = "; C1.display(); C1 = 2.5 + j3.5

cout << "C2 = "; C2.display(); C2 = 1.6 + j2.7

cout << "C3 = "; C3.display(); C3 = 4.1 + j6.2

return 0;

}

## Run time Polymorphism

- Run time polymorphism is achieved when the object's method is invoked at the run time instead of compile time.
- It is achieved by method overriding which is also known as dynamic binding or late binding.

function <sup>overriding</sup> ~~overloading~~ ⇒ function <sup>overriding</sup> ~~overloading~~ occurs when a derived class has a definition for one of the member function of the base class. that base function is said to be overridden.

### Run time Polymorphism Example ⇒

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

class Animal
{
    public
    void eat()
    {
        cout << "Eating....";
    }
};
```

```
Class Dog : public Animal
```

```
{  
    public:  
    void eat()  
    {  
        cout << "Eating bread ---";  
    }  
};
```

```
int main()  
{  
    Dog d = Dog();  
    d.eat();  
    return 0;  
}
```

Output :

Eating bread

Virtual function in C++

- A virtual function is a member function which is declared within a base class and is re-defined (overridden) by a derived class.
- When you refer to a derived class object using a pointer or a reference to the base class, you can call a virtual function for that object and execute the derived class's version of the function.
- Virtual functions ensure that the correct function is called for an object, regardless of the type of reference (or pointer) used for function call.
- They are mainly used to achieve **Runtime Polymorphism**.
- Functions are declared with a virtual keyword in base class.
- The resolving of function call is done at runtime.



## Rules of Virtual functions $\Rightarrow$

- 1  $\Rightarrow$  Virtual function cannot be static.
- 2  $\Rightarrow$  A virtual function can be a friend function of another class.
- 3  $\Rightarrow$  Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.
- 4  $\Rightarrow$  The prototype of virtual functions should be the same in the base as well as derived class.
- 5  $\Rightarrow$  They are always defined in the base class and overridden in derived class.

## Virtual function Program

# include &lt;iostream&gt;

using namespace std;

Class Base {

public:

virtual void print() // virtual function

{

cout &lt;&lt; "print base class \n";

}

void show()

{

cout &lt;&lt; "show base class \n";

}

};

Class Derived : public Base

{

public:

void print()

{

cout &lt;&lt; "print derived class \n";

}

void show()

{

cout &lt;&lt; "show derived class \n";

}

};

```
int main()
{
    Base *bptr;
    Derived d;
    bptr = &d;
    // virtual function, binded at runtime
    bptr -> Print();
    // non-virtual function, binded at Compile time
    bptr -> Show();
    return 0;
}
```

Output =>

Print derived class  
Show base class

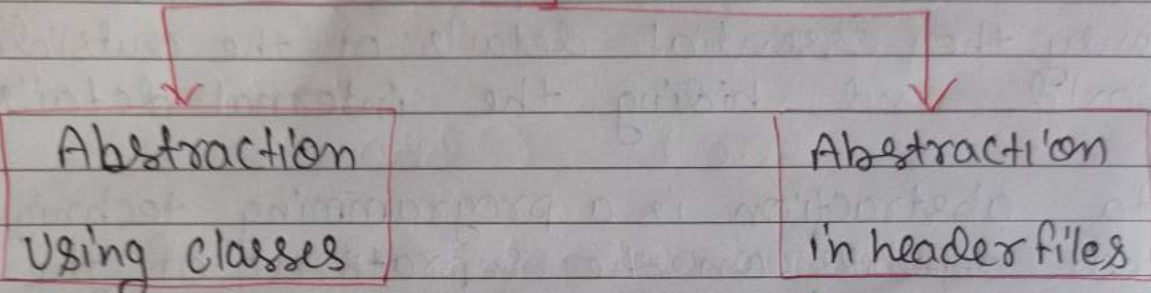
## Data abstraction in C++ / Data hiding.

- Data Abstraction is a process of providing only the essential details of the outside world and hiding the internal details.
- Data abstraction is a programming techniques that depends on the separation of the interface and implementation details of the program.
- A real life example of abstraction is AC, which can be turned ON or OFF, change the temperature change the mode, and other external component such as fan, swing. But, we don't know the internal details of AC, such as how it works internally.
- C++ provide a great level of abstraction for example, `pow()` function is used to calculate the power of a number without knowing the algorithm the function follows.
- In C++ program if we implement class with private and public members then it is an example of data abstraction.

Data abstraction can be achieved in two ways:

- 1) data abstraction using classes
- 2) data abstraction using header files.

Data Abstraction



Data Abstraction Using Classes

- An abstraction can be achieved using classes.
- A class is used to group all the data member and functions into a single unit by using the access specifiers.
- A class has the responsibility to determine which data member is to be visible outside and which is not.

Access Specifiers Implement Abstraction:

- public - specifier  $\Rightarrow$  When the members are declared as public, members can be accessed anywhere from the program.
- Private Specifier  $\Rightarrow$  When the members are declared as private, members can only be accessed only by the member function of the class.

Example data abstraction using classes.

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

```
class Sum
```

```
{
```

private:

int x, y, z; // private variables

public:

void add()

{  
cout << "Enter two numbers";

cin >> x >> y;

z = x + y

cout << "Sum of two no. = " << z;

}  
};

int main()

{  
Sum ob;  
ob.add();

return 0;

}

## Data Abstraction using header files

- Using header files we can hide some data in C++ program. Such as pow()

function available is used to calculate the power of a number without actually knowing which algorithm functions uses to calculate the power. Thus, we can say that header files hides all the implementation details from the user.

Example data Abstraction using header files:-

```
#include <iostream>
#include <math.h>
using namespace std;
```

```
int main()
```

```
{
```

```
    int n = 4;
```

```
    int power = 3;
```

```
    int result = pow(n, power); // pow(4, 3)
```

$4^3 = 64$

```
    cout << "power = " << result;
```

```
    return 0;
```

```
}
```

output  $\Rightarrow$  64