```
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 + j 6.2
    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 ~~overloading~~ overriding ⇒ function ~~overloading~~ overriding 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 ⇒

1 ⇒ Virtual function cannot be static.

2 ⇒ A virtual function can be a friend function of another class

3 ⇒ Virtual functions should be accessed using pointer or reference of base class type to achieve runtime polymorphism.

4 ⇒ The prototype of virtual functions should be the same in the base as well as derived class.

5 ⇒ They are always defined in the base class and overridden in derived class.

Virtual function Program

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

Class Base {

public:
 virtual void print()      // virtual functio
{
 Cout << " print base class \n";
}
 Void show()
{
 Cout << " show base class \n";
}
};

Class Derived : public Base
{
   public:
    void print()
   {
    Cout << " print derived class \n";
   }
    Void show()
   {
    Cout << " show derived class \n";

   }
};
```

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

Output ⇒

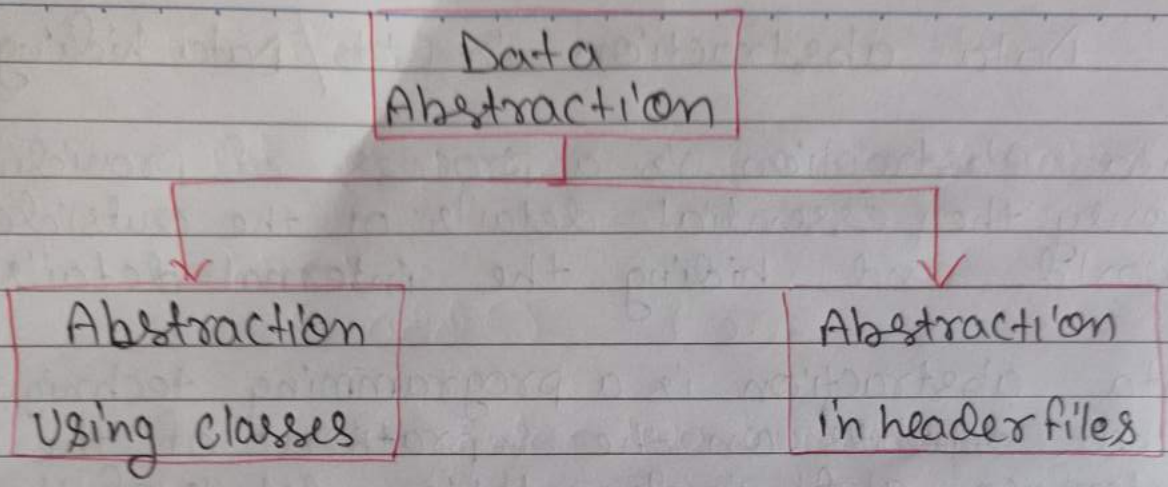
Print derived class
Show base class

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 seprations 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 achived in Two ways:

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


Data Abstraction
Abstraction using classes
Abstraction in header files

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 ⇒ When the members are declared as public, members can be accessed anywhere from the program.
- Private Specifier ⇒ 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 ⇒ 6.4

C++ Tutorials

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

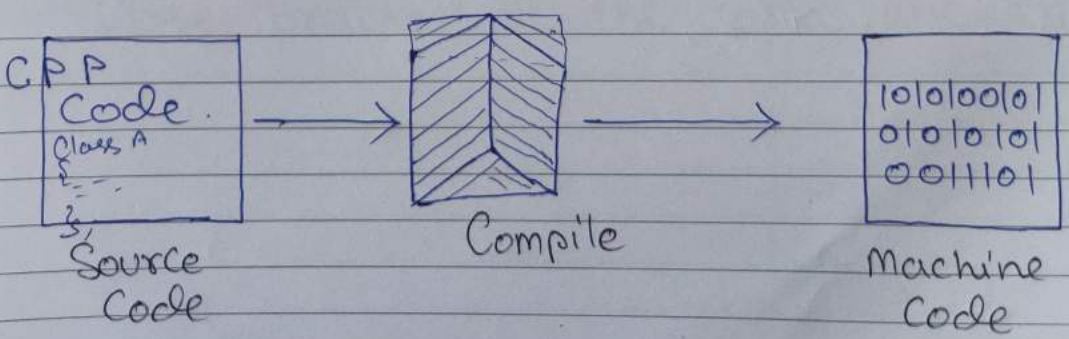
Ans ⇒ C++ is a high-level "Semi-object oriented" programming Language developed by "Bjarne Stroustrup" in 1979 at "Bell Labs."

⇒ C++ is a "general purpose", "Case-sensitive", "free-from" 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

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      } → multi Comment
        Comment */
```

print Statement ← { Cout << " output statement ";

```
// Single Line Comment } → Single Comment

  return value of return;
```

main function return statement

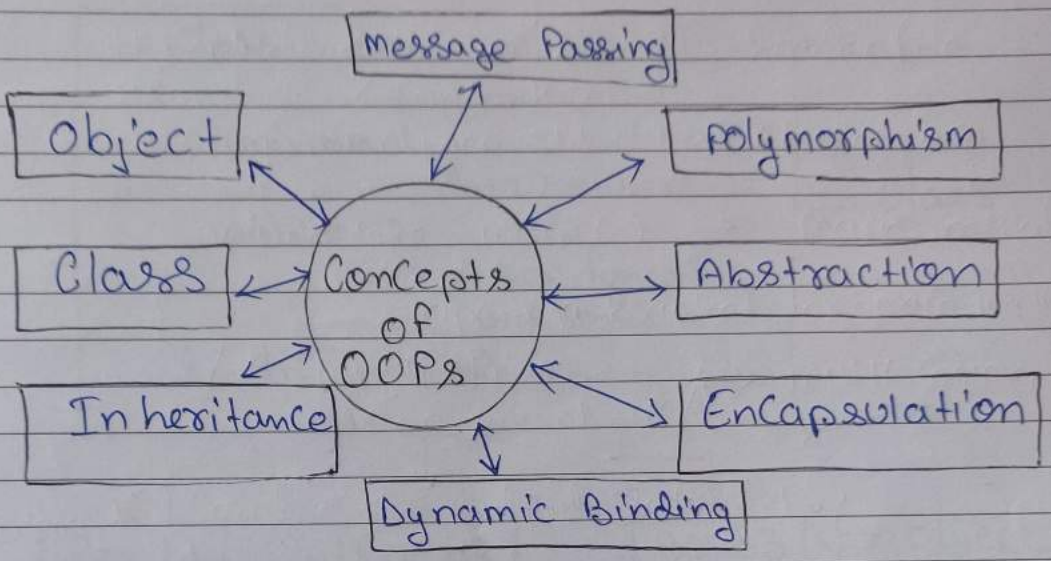
}

Close bracket of the main function

C++ Tutorials

M.I

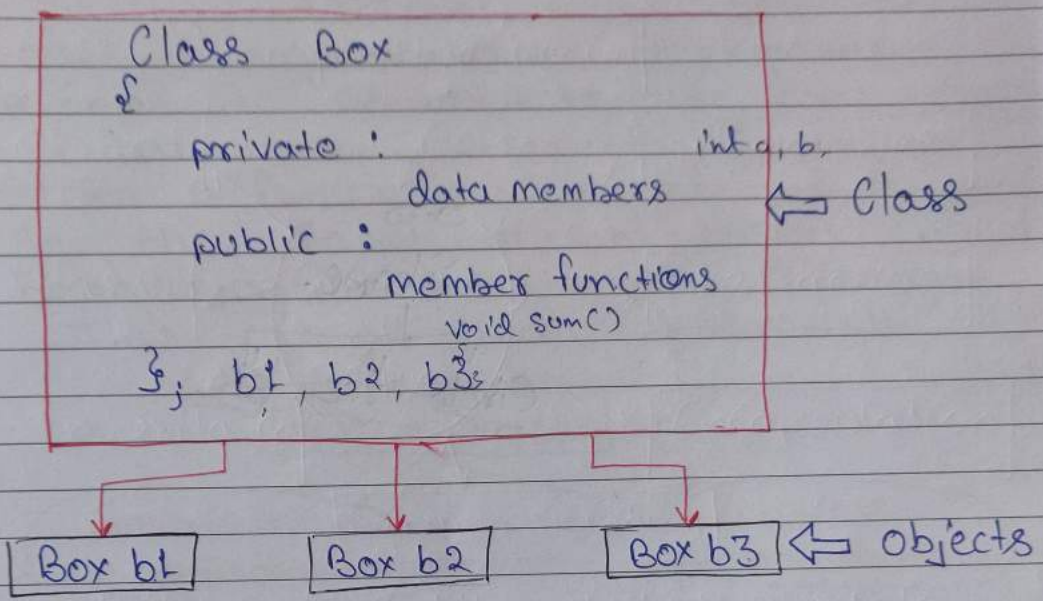
Q ⇒ What is C++ OOPs Concepts 2. Explain.



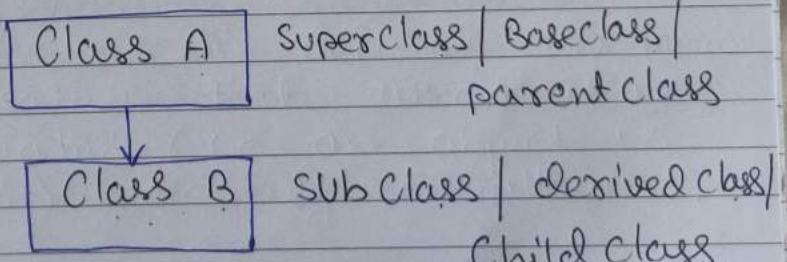
1) Object ⇒ ◆ Any Entity that has State and behavior is known as object.
- ◆ object is a "Active" Entity.
- ◆ An object is an instance of a Class
- ◆ When a Class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated.

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.

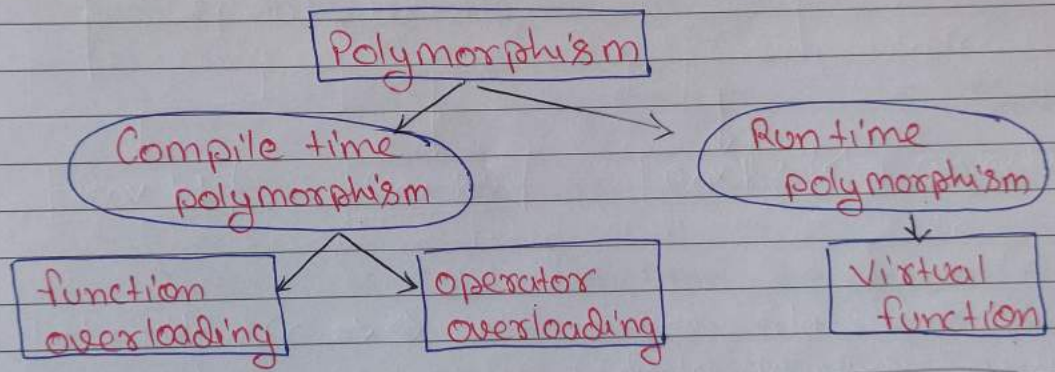


Syntax ⇒

```
Class A
{ --- data member
--  mum f-
};
 Class B : public A
{ ---
};
```

4) Polymorphism ⇒ When one task is performed by different ways are known as polymorphism. poly + morphism

for Ex ⇒ A person at the same time can have different character like a father, a husband and employee. So the same person posses different behaviour in different situations. This is called Polymorphism.



Kailash Joshi

5) Abstraction ⇒ Hiding internal details and showing functionality.

for Ex ⇒ Phone call, we don't know the internal processing.

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

6) Encapsulation ⇒ Binding (or wrapping)

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

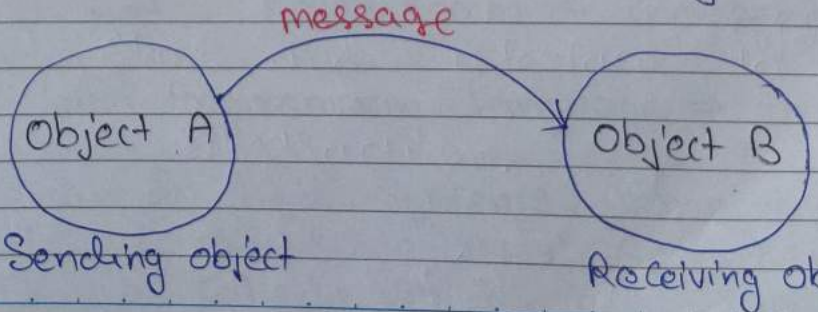
for Ex ⇒ Capsule, It is wrapped with different medicines.

7) Dynamic Binding ⇒ In dynamic Binding, the code to be executed in response to function call is decided at runtime.

- C++ has virtual function to support this.

Kailash Joshi

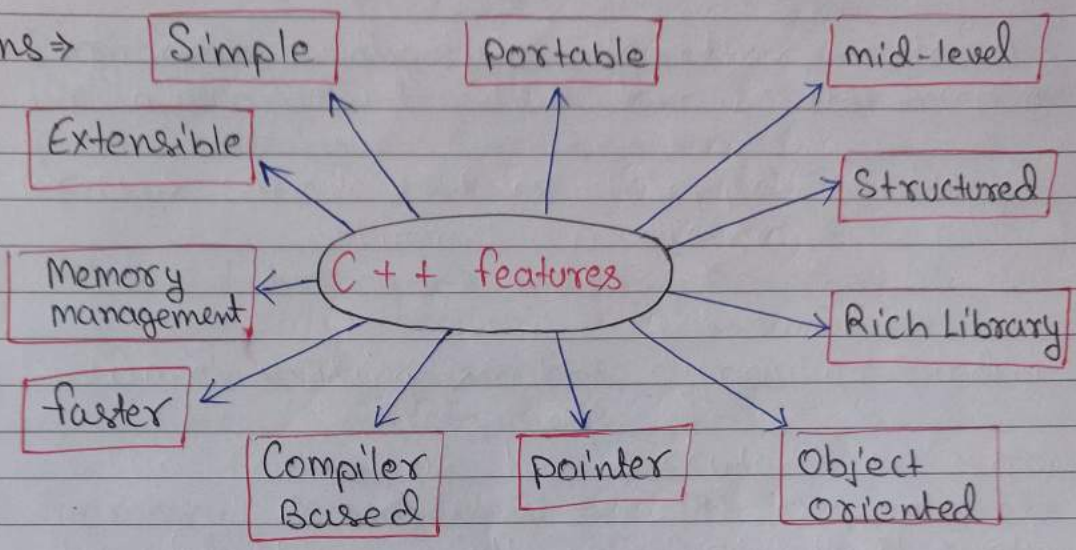
8) Message Passing ⇒ Objects Communicate with one another by sending and receiving information to each other through Message Passing.



C++ Tutorials

Q ⇒ Explain Different features of C++.

Ans ⇒



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.

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

4) Structured Programming Language ⇒ 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 ⇒ C++ provides a lot of inbuilt functions that makes the development fast.

6) Memory Management ⇒ 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 ⇒ The Compilation and execution time of C++ Language is fast.

Kailash Sahu

8) Pointer ⇒ C++ Provides the feature of Pointer. We can directly interact with the memory by using the pointer.

9) Extensible ⇒ It is Extensible because it can easily adopt new features.

10) object-oriented programming ⇒

C++ Tutorials

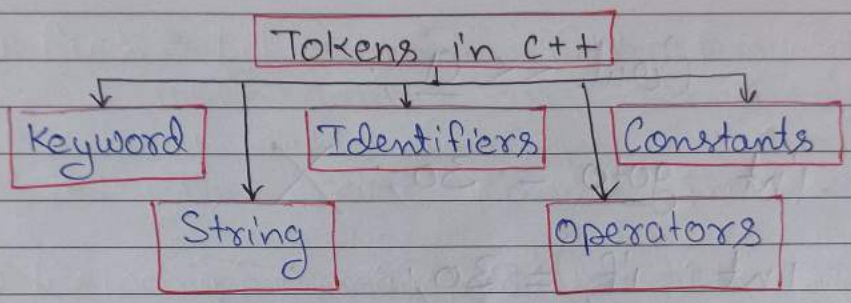
CSE Gyan

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.

• Token act as building blocks of a program.



1) Keywords ⇒ • Keyword 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.

2) Identifiers ⇒ 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 ⇒. 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 Constant
'A'      // Character Constant
```

4) Strings ⇒ • 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 ⇒

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

5) Operators ⇒ C++ operator is a Symbol that is used to perform mathematical or logical manipulations.

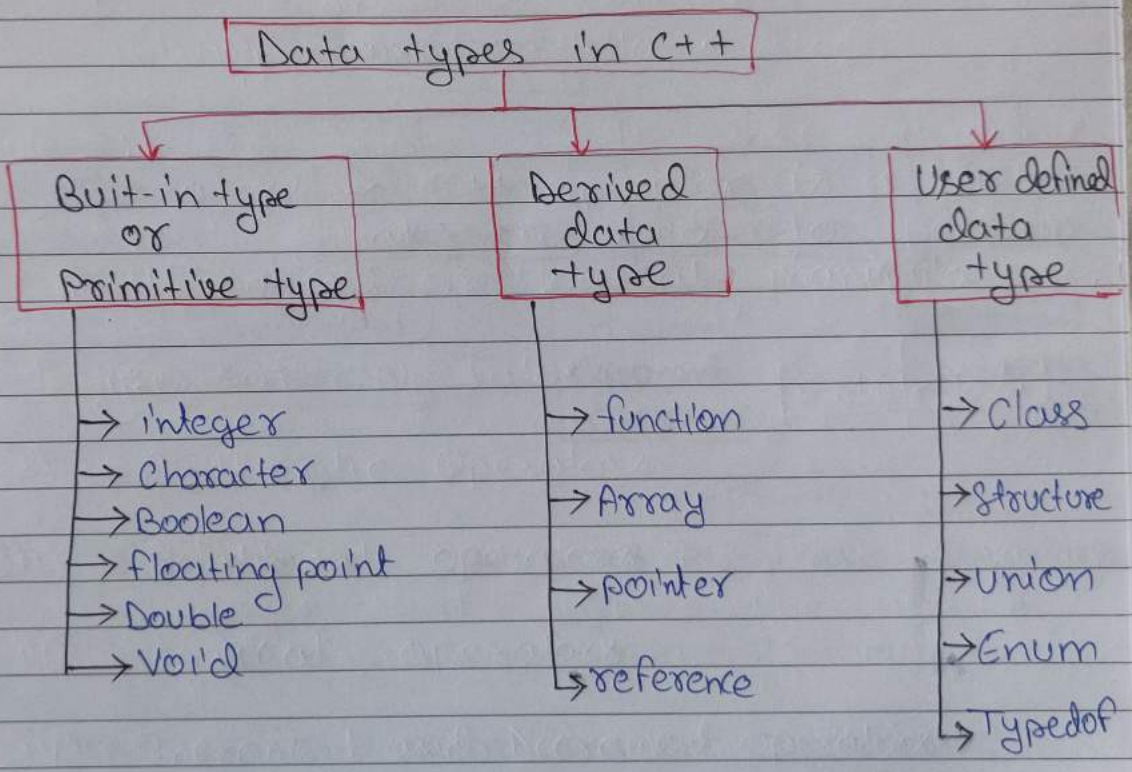
Types of Operators:

Kailash 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++.



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

Kailash Joshi

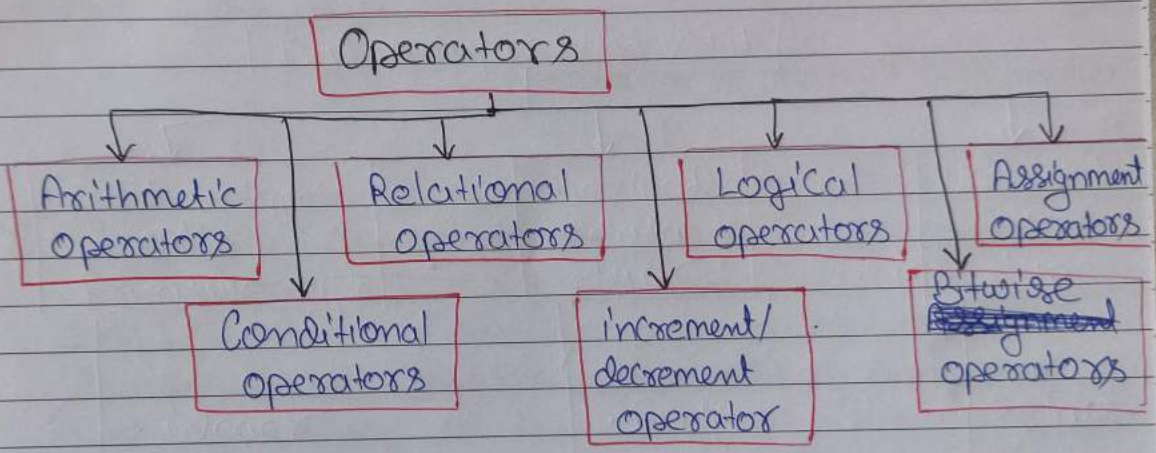
Ex ⇒ sizeof(char) ⇒ 1 (in byte)

C++ Tutorials

Q⇒ What do you mean by operators in C++ ?

Soln⇒ 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 Joshi

1) Arithmetic operators ⇒ These operators are used to perform arithmetic/ mathematical operations on operands

Ex ⇒ +, -, *, /, %

/ → qu

% → Reminder

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

13/2 = 6

13%2 = 1

Camlin

2) Relational operators ⇒ These are used for The Comparison of the values of two operands

Ex ⇒ ==, >=, <=, >, <

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

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

3) Logical operators ⇒ Logical operators are used to Combinning two or more Conditions / Constraints or to Complement the Evalution 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

4) Assignment Operators :⇒ These operator 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 ⇒ a = 10;

a = b

a += b; ⇒ a = a+b

5) Conditional operators :⇒ 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 (? :)

Kailash Jothi

Syntax ⇒

Variable = Expression ? true statement : false statement ;

Ex ⇒

int a = (b>c) ? b : c ;

Camlin

6) Increment / Decrement Operator ⇒ 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 ⇒ x++; // Same as x = x+1

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

Kailash Joshi

7) Bitwise Operators ⇒ 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) ^ // Bitwise XOR operator

4) ~ // 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 → Sort int → int → unsigned int → long int → float → double → long double etc.

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

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

C++ Classes and objects:

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

- 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.
```

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();
```

*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 Joshi

WAP in c++ for more then 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

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;
};
```

Kailash Joshi

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 class definition
    {
        Body of Member function
    }
};
```

Kailash Joshi

2) Outside Class definition ⇒ (methods) ⇒

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 the **scope resolution ::** operator, followed by the name of the function.

NOTE ⇒ You access methods just like you access attributes, by creating an object of the class and using the **dot Syntax (.)**.

Syntax ⇒

```
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.
```

Kailash Joshi

C++ Tutorials

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 Constructor are different from normal member function?

Kailash 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.

Camlin

WAP to Create a Constructor in C++

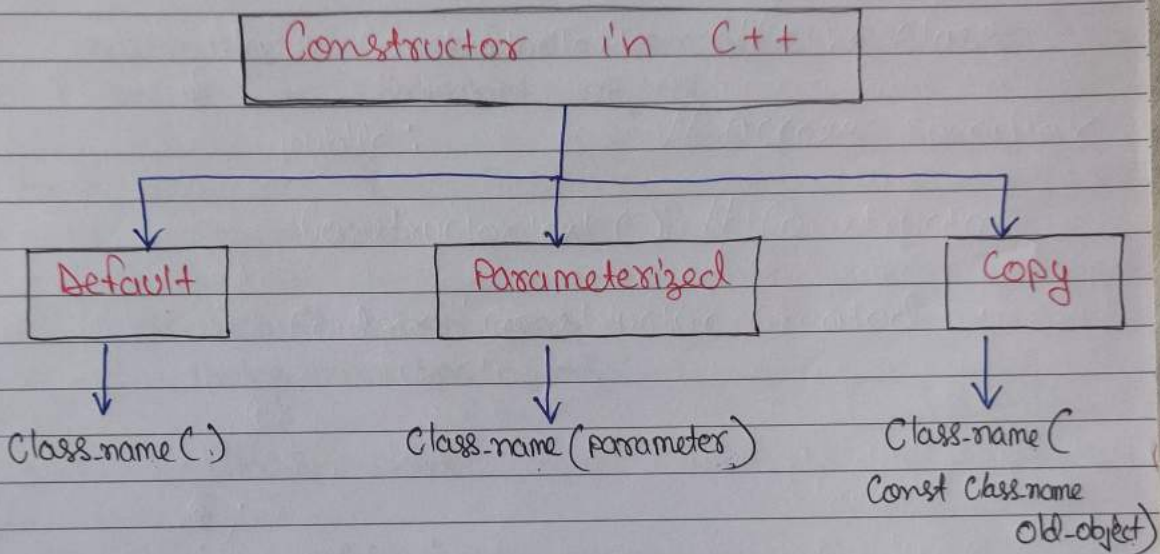
```
#include <iostream>
using namespace std;
Class ConstructorTest.        // The Class
{
    public:                   // access specifier
      ConstructorTest.()      // Constructor
    {
      Cout << "Constructor Created";
    }
};
int main()
{
    ConstructorTest ob;       // Create an
                              object of ConstructorTest
    return 0;                 (this will call the Constructor)
}
```

Kailash Joshi

C++ Tutorials

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;
  }
};
```

C++ Tutorials

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;

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;
}
```

Kailash Joshi

Camlin

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 ⇒ The Compiler defines the default Copy Constructor. If the user defines no Copy Constructor, Compiler Supplies its Constructor.

2) User defined Constructor ⇒ The programmer defines the User-defined Constructor

Syntax of User-define Copy Constructor ⇒

```
Class_name (Const Class_name &old_object);
```

Ex ⇒

```
Class A
{
   A (A &x)  // Copy Constructor
   {
      //CopyConstructor body
   }
};
```


Copy constructor can be called in following ways:

```
A a2(a1)
A a2 = a1
```

} a1 initialises the a2 object.

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;

}
```

C++ Tutorials

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 involved 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.

Kailash Joshi

Syntax ⇒

```
Class Classname
{
  public :
  Classname ( ) // Constructor
  { }
  ~ Classname ( ) // destructor
  { }
};
```

WAP for destructor in C++;

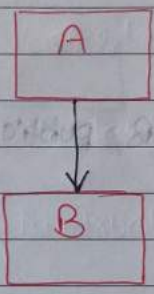
```
#include <iostream>
using namespace std;

Class DistructorTest
{
    Public:
     DistructorTest()    // Constructor
    {
     Cout<<" Constructor Invoked"<<"\n";
    }
     ~DistructorTest()  // destructor
    {
     Cout <<" Distructor Invoked";
    }
};

int main()
{
    DistructorTest ob;

    return 0;

}
```

Kailash Joshi

Inheritance in C++

The capability of a class to drive properties and characteristics from another class is called inheritance.

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



base class / super class / parent class

derived class / subClass / Child Class

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

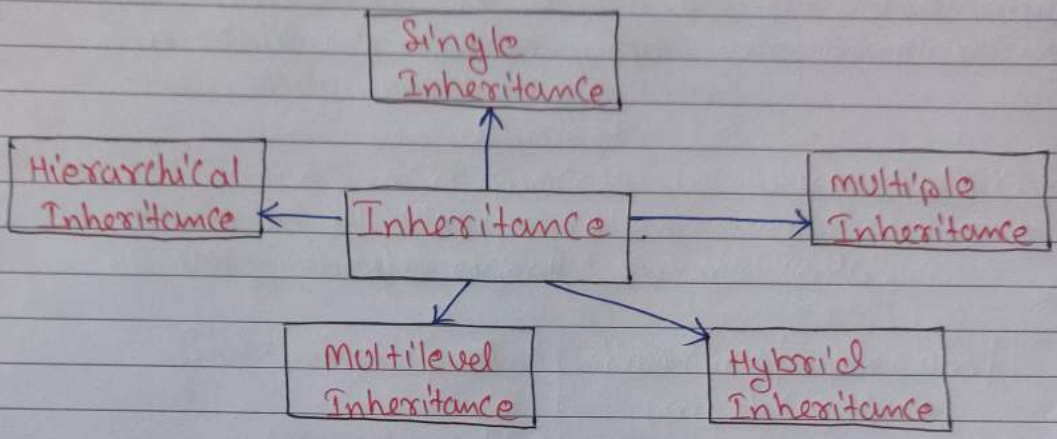
Syntax ⇒ Vailash Joshi

```
Class derived-class-name : visibility-mode base-class-name
{
    // body of the derived class.
}
```

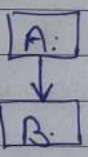
visibility-mode ⇒ The visibility mode specifies whether the features of the base class are publicly inherited or privately inherited. It can be public or private.

C++ Tutorials

Types of Inheritance

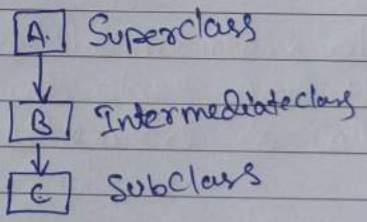


1) Single Inheritance



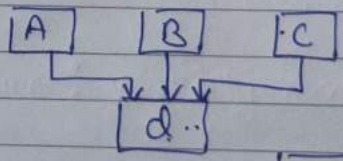

```
Class A
{ ...
};
Class B : public A
{ ...
};
```

2) multilevel Inheritance

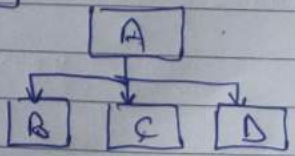


Kailash Joshi

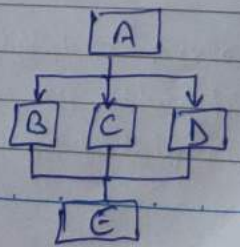
3) multiple Inheritance



4) Hierarchical Inheritance



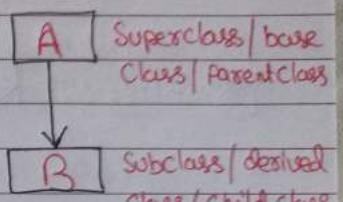
5) Hybrid inheritance



Camlin

C++ Single Inheritance

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



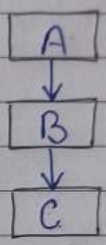
WAP for Single Inheritance

```
#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();
}
```

Kailash 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 <iostream>
using namespace std;
Class A
{
  Public:
  Void display()
   {
    Cout << "super Class display function \n";
   }
};
Class B : public A
{
   Public:
Void show()
   {
   Cout << "Intermediate show function \n";
   }
};
```

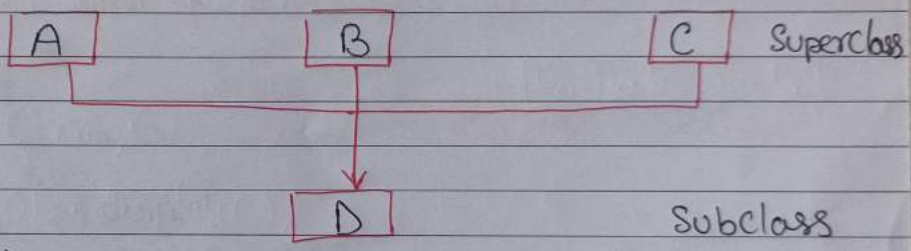
Kailash Joshi

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

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

Kailash Joshi

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 then one Superclass are called multiple Inheritance.



Syntax ⇒

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

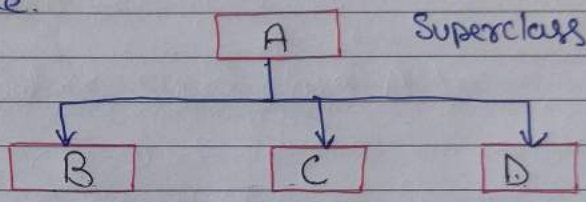
Kailash Joshi

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
  public:                      // Superclass properties
   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;
};

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;

};
```

Kailash Joshi

```
// WAP for Hierarchical Inheritance in C++
#include <iostream>
using namespace std;

Class A                    // Superclass A
{
 public:
 void display()
{
Cout << "Superclass display function \n";
}
};

Class B: public A          // subclass B
{
    public:                                  Kailash Joshi
     void show()
    {
    Cout << " Subclass show method \n";
    }
};

Class C: public A          // subclass C
 {
     public:
     void displayC()
    {
    Cout << " Subclass displayC method \n";
    }
};
```

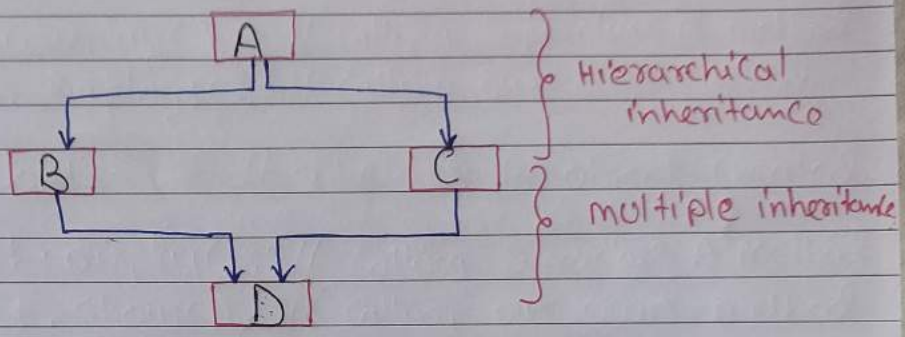
```
int main()
{
    C ob;          // subclass C object Created
    ob.display();  // Calling Superclass A method
    ob.display1(); // Calling our own method.
    B ob1;         // subclass B object Created
    ob1.display(); // Calling Superclass A method
    ob1.show();    // Calling our own method
    return 0;
}
```

Kailash Joshi

Camlin

C++ Hybrid Inheritance

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



Kailash Joshi

Syntax ⇒

```
Class A
{
  Statement;
};

Class B: public A
{
  Statement;
};

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

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";
  }
};
```

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

int main()
{
    D ob;          // Sub-sub Class object Created
    ob.display1();
    ob.show();
    ob.display2();
    B ob1;         // Sub Class object Created.
    ob1.display();
    return 0;
}
```

Kailash Joshi

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 ⇒

```
# 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;
}
```


Camlin

C++ Tutorials

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)
     {
```


Camlin

```
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

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 is 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 is 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 datatype consisting of named values like elements, members etc. that represent integral constants.
- It provides a way to defined 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.I

C++ 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 involved 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.

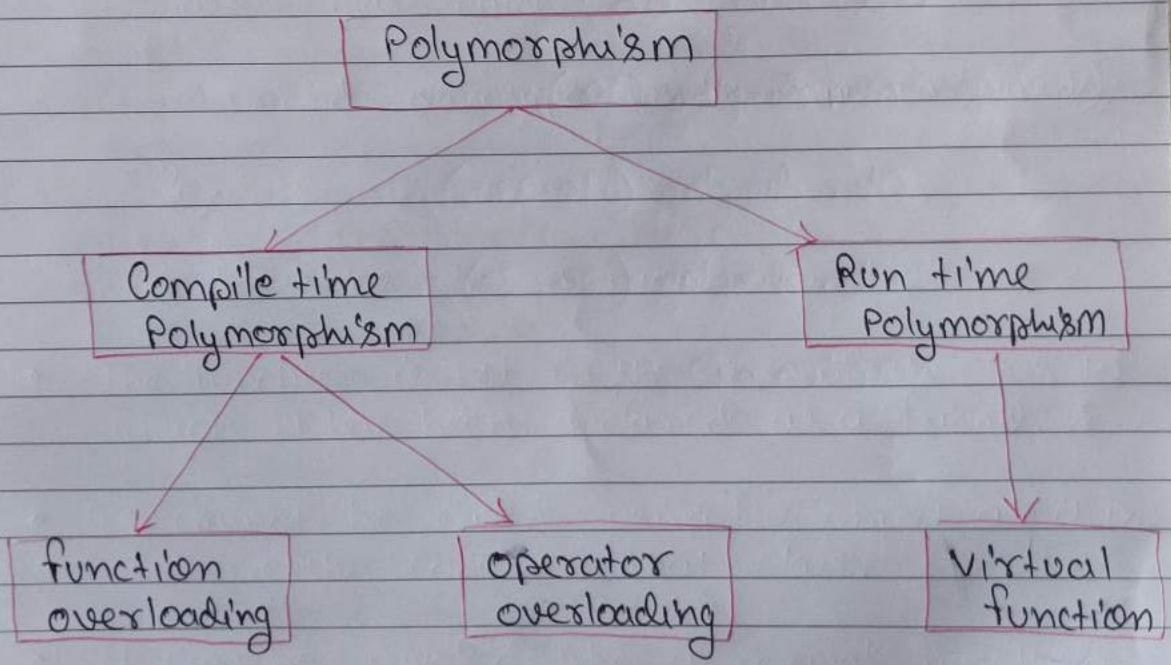
C++ friend function Example

```
#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;
}
```

M.M.I

C++ Polymorphism

- Polymorphism means "many forms" and it occurs when we have many classes that are related to each other by inheritance.
- Paymorphism is the combination of "Poly" + "morphs" 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 posses different behavior in different situations. This is called polymorphism.

Compile time Polymorphism

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

function overloading ⇒ • 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 ⇒

```
# 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 use-define 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(argument-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.

```
// 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()
{
```

```
    cout << real << "+i" << imag;
  }
};
int main()
{
  Complex C1(10,5), C2(2,4);
  Complex C3 = C1+C2;
  C3.print();
}
```

10+12 10+5i
5+4 2+4i

output = 12 + i9

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 ⇒ 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 minus
};
```

```
Void space :: getdata (int a, int b)
{
    x = a;
    y = b;
}
Void space :: display ()
{
    Cout << x;
    Cout << y;
}

Void space :: operator -()
{
    x = -x;
    y = -y;
}

int main()
{
    space S;
    S.getdata (10, -20);
    Cout << S.display();

    -S; // Activates operator -() function

    Cout << S.display();
    return 0;
}
```

10
-20
-10
20

Binary Operator Overloading

- those operators which operate on two operands or data are called binary operators.
- In Binary operator overloading function, there should be one argument to be passed. It is overloading of an operator operating on two operands.

Example of Binary operator (+) overloading.

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

Class Complex
{
   float x;          // real part
   float y;          // imaginary part
public:
   Complex()         // Constructor 1
   {
   }
Complex(float real, float imag) //Constructor 2
{
   x = real;
   y = imag;

}
Complex operator+(Complex);
 void display(void);
};
```