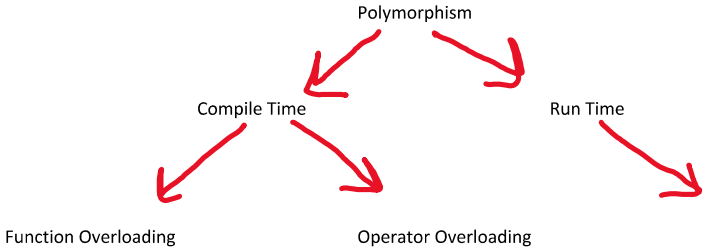
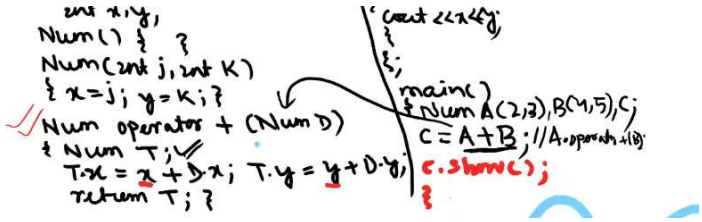
Object Oriented Programming

Thursday, July 29, 2021 11:47 AM

1. An object consist of Name, State and Behavior.
   a. Name
      It is a unique identity for representing an object of a class
   b. State
      It is the representation of attributes of a particular object
   c. Behavior
      It is the set of allowed operations, function or methods
2. Characteristic features of Object Oriented Programming
   a. Emphasis is on Data rather than procedure
   b. Programs are divided into what are known as objects
   c. Data Structures are designed such that they characterize the object
   d. Function are used to manipulate the data of an object (Each object contain data and function to manipulate those data)
   e. Data is hidden and cannot be accessed by external functions
   f. Objects may communicate with each other through functions
   g. New data and function can be easily added whenever necessary
   h. It follows bottom-up approach in program designing
3. #include <iostream>
   Using namespace std;
   a. Using and namespace are the keywords
   b. The statement tells the compiler to use STD namespace, this will bring all the identifiers defined in std to the current global scope
   c. That means a namespace creates a declarative region in which various programming elements can be placed
   d. Std is the namespace in which the entire standard Cpp library is declared
   e. Std namespace simplify access to the standard library
4. Concept Of OOP
   a. Object
   b. Class
      a. A class is a collection of similar type of objects
      b. Class is an user defined datatype
      c. Objects are the variables of class type
      d. The general syntax to define a class is-->
         class Class_Name{
         access Specifier
            Data
            Function
         }
      e. The general to create an object is
         Class_Name Object_Name;
   c. Data Abstraction And Encapsulation
      a. Encapsulation
         i. Rapping up of Data and functions into a single unit (class) is known as encapsulation
         ii. **Data Hiding:** The insulation of the data from direct access by the program is called data hiding, the data is not accessible to the outside vault and only those functions which are wrapped in the class can access it
         iii. **Abstraction:** it refers to the act of representing essential features without including the background details. It is of two types,
            1. Data Abstraction : It means hiding the details about the data
            2. Control Abstraction : It means hiding the implementation detail

      (Encapsulation and Data Hiding is the way to implement Data Abstraction in C++)
5. Inheritance
   a. It is the process by which objects of one class acquire the property of object of another class in OOP the concept of inheritance provides the idea of reusability
   b. This means that we can add additional features to an existing class without modifying it
   c. This is possible by deriving a new class from a existing class
   d. The exsisting class is called as base class and new class is called derived class
   e. The derived class will have the combined features of both the classes
6. Polymorphism
   a. It is the ability to take more than one form



b. 

c. The process of making an operator to exabit different behaviors in different instances is known as operator overloading.

7. Function Overloading
   a. Overloading refers to the use of same thing for different purposes
   b. Using a single function name and different types of arguments and number of arguments to perform different types of task is know as function overloading
   c. Function overloading doesn't depend upon return type of function

```
//<===CODE=====>
#include <iostream>
using namespace std;
Int add(int, int);          //Prototype
float add(float, float); //Prototype
Int add(int, int, int);    //Prototype
Int main(){
        cout<<add(10,10);
        cout<<add(10,6,9);
        cout<<add(2.1,1.4);
        Return 0;
}
Int add(int a, int b){
        Return b + a;
}
float add(float a, float b){
        Return b + a;
}
Int add(int a, int b, int c){
        Return b + a + c;
}

//<===OUTPUT=====>
// 20
// 25
//3.5
```

8. Dynamic Binding
   a. It refers to the linking of procedure or function call to the code to be executed in response to the function call
   b. Dynamic binding is the process of linking a function call to the actual code (function definition) at runtime
   c. The concept of dynamic binding is implemented with the help of (inheritance and runtime polymorphism)
9. Message Passing

```
//<=== CODE =====>
#include <iostream>
Using namespace std;
Class A{
Int b;
Public:
        Void input(int c){
        b=c;
        Cout<<b<<endl;
        }
};
Int main(){
        A ob;
        ob.input(3);
        Return 0;
}
//<===OUTPUT=====>
// 20
// 25
//3.5
```

   a. Message passing involves specifying the name of the object , the name of the function within bracket message and the information to be sent
   b. An object of a class can communicate with object of another class by invoking appropriate member function

If a member function does not alter or modify any data in the class then we may declare it as constant member function using const keyword

The qualifier const is appended to the function prototype and definition.

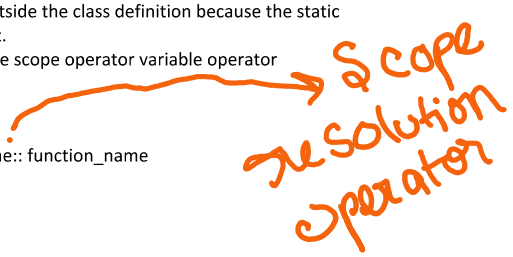
The compiler will generate an error message if such functions try to alter the data values

Static Data Member

Data member of a class can be qualified as static

Characteristics of static data member ->

1 It is initialized to zero, when the first object of it's class is created.
2 Only one copy of that member is created for the entire class and is shared by all the objects of that class
3 It is visible only with in the class
4 Static variables are normally used to maintain common to the entire class
5 (Important) It is mandatory to define each static data member outside the class definition because the static data members are stored separately rather than as part of an object.
6 The general syntax to define static operator is datatype class name scope operator variable operator



Static Member Function

It can me public as well as it can be private

a. It can access other static members of the class
b. The static members function should be called using class_name:: function_name

1. **Static Public Member function**

```
#include<iostream>
usingnamespacestd;
classA{
staticintP;
intb;
public:
voidinput(){
cin>>b;
P++;
}
staticvoidDisplay();//StaticFunction
};
intA::P;//definitionofstaticdatamember
voidA::Display(){
//cout<<b<<endl;
cout<<P<<endl;
}
intmain(){
Aob;
ob.input();
A::Display();
return0;
}
```

2. **Static Private Member function**
   a. WAP to declare global , static and local variable with same name and display the corresponding values on the screen.

```
#include<iostream>
usingnamespacestd;
classB{
private:
staticintc;//staticdatadeclaration
staticvoidcount(){
c++;
}
public:
staticvoidoutput(){
count();
cout<<"c="<<c<<endl;
}
};
intB::c=5;//definitionofstaticdatamember

intmain(){

B::output();
B::output();
ob.input();
return0;
}
```

   3. **Static Vs Global Vs Local**

```
#include<iostream>
Using namespacestd;
intB=1;
classA{
Static int B;
public:
Static void output(){
cout<<"static B="<<B<<endl;
}
};
Int A::B=4;
Int main(){
A ob;
Int B=10;
A::output();
cout<<"globalB="<<::B<<endl;
cout<<"localB="<<B<<endl;
return0;
}
```

3. **OBJECT AS ARGUMENT**
   1. **Call By Value**
   2. **Call By Address**
   3. **Call By Reference**

3. **Call By Reference**

```
#include<iostream>
Using namespace std;
Class S{
Int a,Sum;
public:
Void input (int P){
a=P;
}
Void Display(S,S);
Void Display(S*,S*);
Void Displayr(S&,S&);
};
Void S :: Display(Sob1,Sob2){
Sum=ob1.a+ob2.a;
cout<<"SUN="<<Sum<<endl;
}
Void S::Display(S*ob1,S*ob2){
Sum=ob1->a+ob2->a;
cout<<"SUN="<<Sum<<endl;
}
Void S::Displayr(S&ob1,S&ob2){
Sum=ob1.a+ob2.a;
cout<<"SUN="<<Sum<<endl;
}
Int main(){
S ob1,ob2,ob3;
ob1.input(5);
ob2.input(7);
ob3.Display(ob1,ob2);
ob3.Display(&ob1,&ob2);
ob3.Displayr(ob1,ob2);
Return 0;
}
```

4. **Friend Function**
   1. Cpp allows a mechanism in which a non member function has access permission to the private members of the class.
   2. This can be done by declaring a non-member function friend to the class whose private data is to be accessed.
   3. The friend function declaration is done inside the class either in private or public access specifier.

```
#include<iostream>
Using namespace std;
Class A{
Int P;
Friend void Output(A);//declaration
public:
Void input(int x){
P=x;
}
};
Void Output(A ob){
cout<<"Value of A is:"<<ob.P;
}
Int main(){
A ob;
ob.input(6);
Output(ob);
Return 0;
}
```

   4. WAP to Exchange values between two classes using friend function

```
#include<iostream>
usingnamespacestd;
classB;
classA{
intP;
friendvoidSwap(A&,B&);//declaration
public:
voidinput(intx){
P=x;
}
voidshow(){
cout<<P<<endl;
}
};
classB{
intQ;
friendvoidSwap(A&,B&);//declaration
public:
voidinput(intx){
Q=x;
}

voidshow(){
cout<<Q<<endl;
}
};
```

```
voidSwap(A&ob,B&ob1){
inttemp=ob.P;
ob.P=ob1.Q;
ob1.Q=temp;
}
intmain(){
Aob;
Bob1;
ob.input(6);
ob1.input(7);
ob.show();
ob1.show();
Swap(ob,ob1);
cout<<"Changed"<<endl;
ob.show();
ob1.show();
return0;
}
```

5. WAP to declare friend function in two classes to calculate the sum of integers of both the classes

```
#include<iostream>
usingnamespacestd;
classB;
classA{
intP;
friendintSum(A&,B&);//declaration
public:
voidinput(intx){
P=x;
}
voidshow(){
cout<<P<<endl;
}
};
classB{
intQ;
friendintSum(A&,B&);//declaration
public:
voidinput(intx){
Q=x;
}

voidshow(){
cout<<Q<<endl;
}
};
intSum(A&ob,B&ob1){
returnob.P+ob1.Q;
}
intmain(){
Aob;
Bob1;
ob.input(6);
ob1.input(7);
cout<<"Added:"<<Sum(ob,ob1)<<endl;

return0;
}
```

6. Friend Class
   i. An entire class can be declared as friend class when all the functions made to access another class.
   ii. The friend is not transferable or inheritable from one class to another.
   iii. Declaring class A to be friend of class B does not mean that, class B is also friend of class A, i.e. friendship is not exchangeable.
   iv. The friend class are applicable when we want to make available private data of a class to another class.
   v. WAP to swap values using friend class

```
#include<iostream>
usingnamespacestd;

classswapInt;
classInteger{
intx,y;
public:
voidinput(inta,intb){
x=a;y=b;
}
voiddisplay(){
cout<<"x="<<x<<"y="<<y<<endl;
}
friendclassswapInt;
};
classswapInt{
public:
voidSwap(Integer&I){
inttemp=I.x;
```

```
I.x=I.y;
I.y=temp;
}
};
intmain(){
Integerob1;
ob1.input(1,2);
ob1.display();
swapIntob2;
ob2.Swap(ob1);
ob1.display();
}
```

vi. WAP to declare 3 classes, declare integer array as data member in each class. Perform addition of two data member arrays into array of third class.

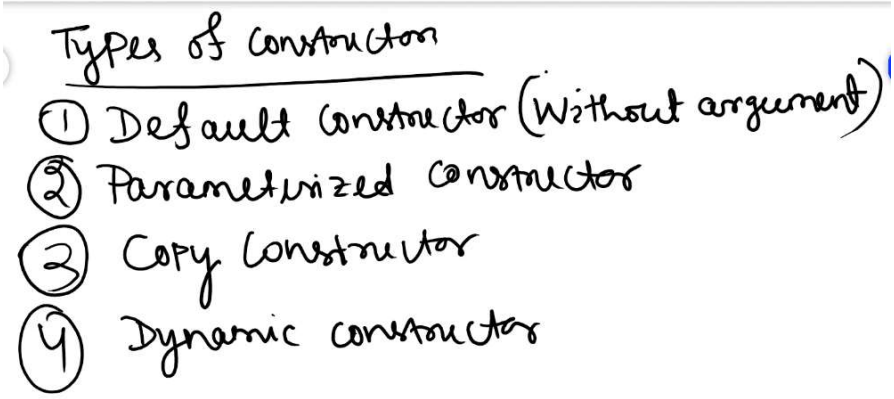
```
#include <iostream>
using namespace std;

class B;
class C;
class A
{
   int x[5];

public:
   void input()
   {
      cout << "Enter the integers : ";
      for (int i = 0; i < 5; i++)
      {
         cin >> x[i];
      }
   }
   friend C output(A, B, C);
};

class B
{
   int x[5];

public:
   void input()
   {
      cout << "Enter the integers : ";
      for (int i = 0; i < 5; i++)
      {
         cin >> x[i];
      }
   }
   friend C output(A, B, C);
};

class C
{
   int x[5];

public:
   friend C output(A, B, C);
   void show()
   {
      for (int i = 0; i < 5; i++)
      {
         cout << x[i] << " ";
      }
   }
};

C output(A a, B b, C c)
{
   for (int i = 0; i < 5; i++)
   {
      c.x[i] = a.x[i] + b.x[i];
   }
   return c;
}

int main()
{
   A a;
   B b;
   C c;
   a.input();
```

```
    b.input();
    c = output(a, b, c);
    c.show();

    return 0;
}
```

vii. WAP to perform addition of two complex numbers using print function and the concept of returning object.

5. Constructor
   i. When a data object is created it's data member contain garbage value . Hence to initialize the data member, cpp use special member function constructor
   ii. Constructor constructs the object destructor destroys the object
   iii. The compiler automatically executes this constructor function and destructor function
   iv. When an object of the class is created the constructer function is called by the compiler
   v. It is optional to declare constructor and destructor
   vi. If the programmer does not define them the compiler execute implicit constructor and destructor
   vii. Both constructor and destructor are special member function where the constructor name is as the class it belongs to.
   viii. Destructor name is also same as class name preceded by ~ tilde sign
   ix. WAP to define a constructor and initialize class data member using that

```
#include <iostream>
Using namespace std;
Class NUM{
        Int a,b,c;
        Public:
                Int x;
                Void output(){
                        Cout<<a<<b<<c<<x;
                };
                NUM(); /// Constructor definition
};
NUM::NUM(){
        a=2;
        b=3;
        c=5;
        x=4;
        Cout<<"Constructor is called";
}
Int Main(){
       NUM N1;
        NUM N2;
        N1.output();
        Return 0;
}
```



6. Con

```
Class A{
        Int a;
        Float b;
        Char c;
        Public:
                Void output(){
                        Cout<<a<<b<<c;
                }
                A(); // DEfault
                A(int float);
                A(int , float, char);
}

A::A(int p, float q, char r){
        A=p;
        B=q;
        C=r;
```

```
}

A::A(int p, float q){
        A=p;
        B=q;
        C='';
}

A::A(){
        A=NULL;
        B=NULL;
        C=NULL;
}
int main(){
        A ob1(1,2.2,'a');
        Ob1.output();
        A ob2(1,2.2);
        ob2.output();
        A ob3;
        Ob3.output();
}
```

7. Constructor with default argument

```
#include<iostream>
#include<cmath>
usingnamespacestd;

classPower{

intnum;
intresult;
intP;
public:
Power(int,int);
voidshow(){
cout<<num<<"^"<<P<<endl;
}
};

Power::Power(intn=3,intp=4){
num=n;
P=p;
result=pow(n,P);
}
intmain(){
PowerP1;
P1.show();
PowerP2(5);
P2.show();
PowerP3(6,7);
P3.show();
return0;
}
```

8. Copy Constructor
   i. It is possible to pass reference of an object to a constructor such type of constructor are known ac copy constructor

```
#include<iostream>
usingnamespacestd;

classNum{
intn;
public:
Num(){//DEfault

}
Num(intk){//Parameterised
n=k;
}
Num(Num&ob){//Copy
n=ob.n;
}
voidshow(){
cout<<"n="<<n<<endl;
}
};
Int main(){
NumN1(50);//parameterizedconstructor
N1.show();
NumN2(N1);//copyconstructor
N2.show();
NumN3=N2;//copyconstructor
N2.show();
NumN4;//defaultconstructor
N4=N1;//constructoroverloading
return0;
```

```
}
```

9. Destructor
   i. It is used to destroy the object that have been created by a constructor.
   ii. Destructor is a special member function whose name is same as a class name by is preceded by a tilde (~) operator.
   iii. Characteristics of Destructor
      - Link a constructor the destructor does not have any return type nor even void.
      - It's name is same as a class name by is preceded by a tilde (~) operator.
      - The destructor doesn't have any argument
      - **The destructor neither have default values nor can be overloaded**
   iv. When is destructor called?
      - A destructor function is called when the object goes out of the scope
        - Function Ends
        - Program ends
        - Block containing local variable ends
        - When a delete operator is called
      - If the class name is A, then the syntax of destructor is ~A();
      - There can only be one destructor in class
   v. When do we need to write user defines destructor?
      - If we do not write our own destructor in the class the compiler calls the implicit destructor.
      - The default destructor works fine unless we have dynamically allocated memory in a class.
      - When a class contains a pointer to a memory allocated in a class we should write a destructor to release the memory before the class instance is destroyed.
   vi. Example Of Destructor

```
#include <iostream>
using namespace std;
class A{
	public:
		A(){
			cout<<"constructor called";
		}
		~A(){
			cout<<"Destructor called";
		}
};
int main(){
	A ob;
	return 0;
}
```

10. Dynamic constructor
   i. Dynamic constructor is used to allocate memory to the object at the runtime using new operator

```
#include <iostream>
using namespace std;
class D{
	int *p;
	public:
		D(){
			cout<<"Constructor here"<<endl;
			p=new int;
			*p=5;
		}
		D(int x){
			cout<<"Dynamic Constructor here"<<endl;
			p=new int;
			*p=x;
		}
		int output(){
			return *p;
		}
		~D(){
			cout<<"Destructor here"<<endl;
			delete p;
		}
};
int main(){
	D ob;
	D ob1(3);
	cout<<"ob : "<<ob.output()<<endl;
	cout<<"ob1 : "<<ob1.output()<<endl;
	return 0;
}
```

   ii. WAP to create as class NUM with data member size of the array and a pointer to the array. Initialize data of array via new operator, and show addition of all elements in array.

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

Class C;
Class A{
	Int x[10]={0,1,2,3,4,5,6,7,8,9};
```

```
        int x[10]={0,1,2,3,4,5,6,7,8,9};
        public:
            Void display(){
                Int i=0;
                while(i<10){
                cout<<x[i]<<"";
                i++;
            }
        }
        friendclassC;
};
classB{
        inty[10]={0,1,2,3,4,5,6,7,8,9};
        public:
            voiddisplay(){
                inti=0;
                while(i<10){
                cout<<y[i]<<"";
                i++;
            }
}
friendclassC;
};
classC{
intz[10]={0,1,2,3,4,5,6,7,8,9};
public:
voidAdd(A&a,B&b){
inti=0;
while(i<10){
z[i]=a.x[i]+b.y[i];
i++;
}
}
voiddisplay(){
inti=0;
while(i<10){
cout<<z[i]<<"";
i++;
}
}
};
intmain(){
Aa;
Bb;
Cc;
a.display();
cout<<endl;
b.display();
cout<<endl;
c.Add(a,b);
c.display();
return0;
}
#include<iostream>
#include<cstring>

usingnamespacestd;

classst{
char*s;
intsize;
public:
voiddisplay(){
cout<<s<<endl;
}
st(char*);
~st();
};

st::st(char*c){
size=strlen(c);
s=newchar[size+1];
strcpy(s,c);
}

st::~st(){
delete[]s;
}
intmain(){
stob1("xyz");
ob1.display();
return0;
}
```

11. This Pointer

```
#include<iostream>
```

```
usingnamespacestd;

classNum{
intn;
public:
voiddisplay(){
cout<<"minimumvalueis:"<<n<<endl;
}
voidinput(){
cout<<"Enterthevalueofn:";
cin>>n;
}
Nummin(Numob){
if(ob.n<n){
returnob;
}else{
return*this;
}
}
};


intmain(){
Numob1;ob1.input();
Numob3;ob3.input();
Numob2=ob1.min(ob3);
ob2.display();
return0;
}
```

12. When Constructor and Destructors are private they cannot be executed implicitly by compiler and hence it is compulsory to execute them explicitly

```
#include<iostream>

usingnamespacestd;

classA{
intx;
A(){
x=7;
cout<<"constructorcalled";
}
~A(){
cout<<"destructorcalled";
}
public:
voidshow(){
this->A::A();//invokesconstructor
cout<<"x="<<x;
this->A::~A();//invokesdestructor
}
};


intmain(){
A*p;
p->show();
return0;
}
```

13. Constant Object
    i. Const AB ob(5);
    ii. only constructor can initialize constant object
    iii. Constant object data member are called as read only data member because their values cannot be changed
    iv. Constant can access only constant functions.

```
#include<iostream>

usingnamespacestd;

classAB{
intx;

public:
AB(intm){
x=m;
}
voidshow()const{
cout<<"a="<<x;
}
};


intmain(){
constABob(3);
```

```
ob.show();
return0;
}
```

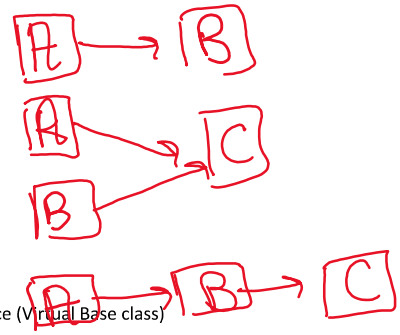
14. Inheritance
    i. The procedure of creating a new class from one or more existing classes is known as inheritance
    ii. the existing class in called as base or parent class and the new class is called as derive class, sub class or child class
    iii. Single Inheritance

    iv. Multiple Inheritance

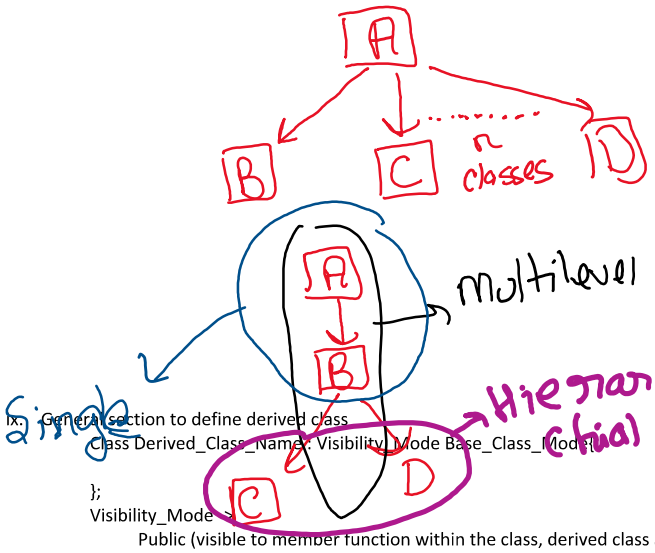
    v. Multilevel Inheritance

    vi. Hierarchical Inheritance

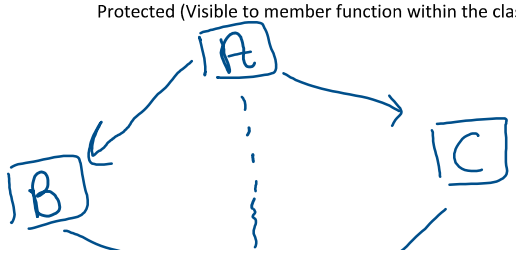
    vii. Hybrid Inheritance

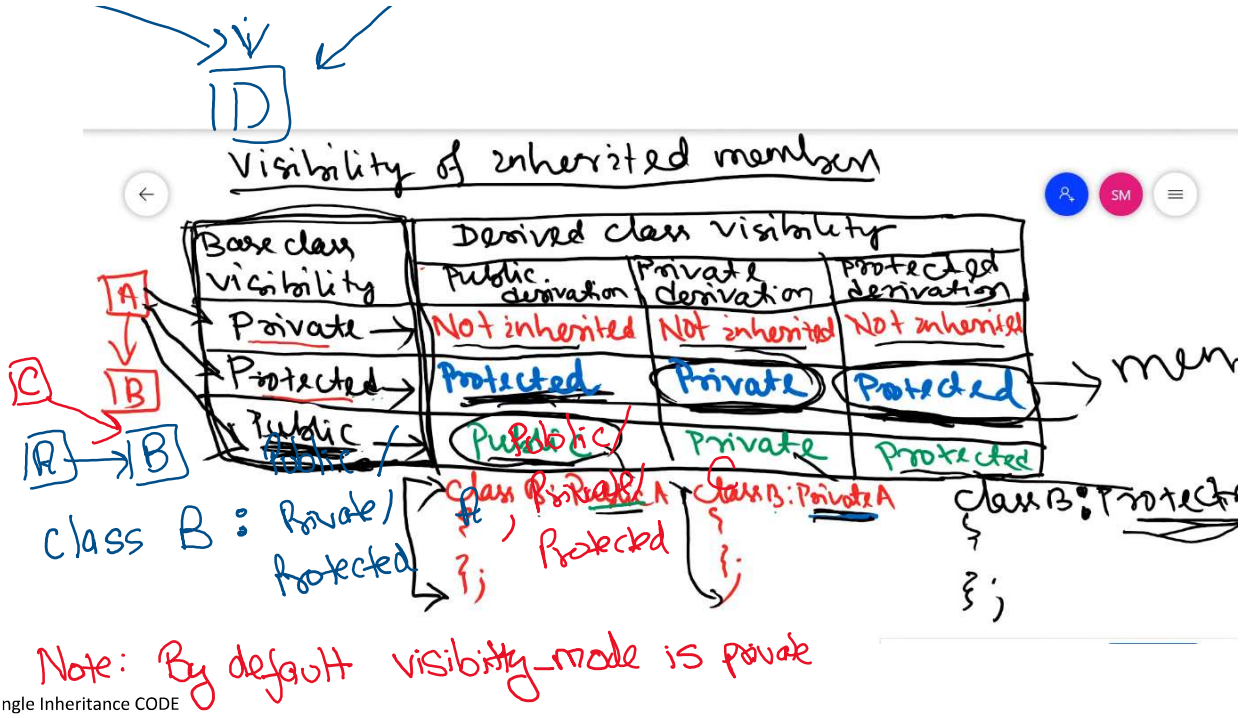


    viii. Multipath Inheritance (Virtual Base class)



    ix. General section to define derived class
        Class Derived_Class_Name: Visibility_Mode Base_Class_Mode{
        };
        Visibility_Mode ->
            Public (visible to member function within the class, derived class and through object),
            Private (Visible to member function within the class but not in derived class),
            Protected (Visible to member function within the class and derived class)





x. Single Inheritance CODE

```
Class A{
        Public:
                Int x;
};
Class B:public A{
        Public:
                Int y;
};
Int main(){
        B ob;
        Ob.x=10;
        Ob.y=20;
        Cout<<ob.x<<ob.y<<endl;
}
```

xi. Multilevel Inheritance

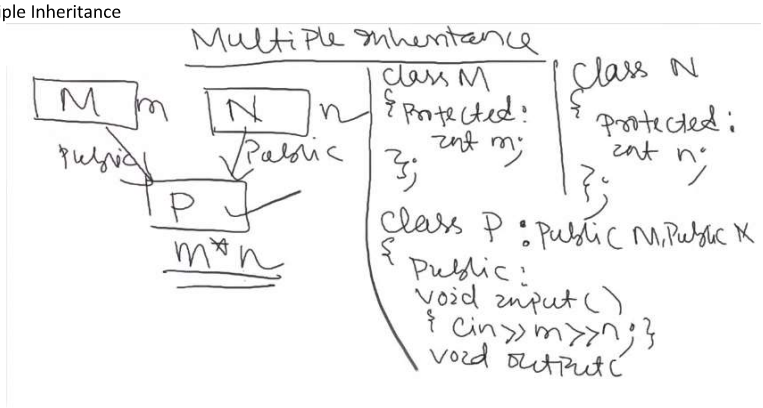
```
Class A{
        protected:
                Int a;
        Public:
                Void input1(int x){
                        A=x;
                }
                Void display1(){

                        Cout<<a<<endl;
                }
}
Class B:public class A{
        Protected:
                Int b;
        Public:
                Void input2(int x){
                        b=x;
                }
                Void display(){
                        Cout<<b<<endl;
                }

}
Class C:public class C{
        int c;
        Public:
                Void input(int x){
                        b=x;
                }

                Void show(){
                        Cout<<c<<endl;
                        Display();
                        Display1();
                }
```

```
}
Int main(){
        C ob;
        Ob.input(1);
        Ob.input2(4);
        Ob.input1(7);
        Return 0;
}
```

xii. Create a class student with data member roll number, derive a class test form class student with data member subject1 and subject 2. derive a class result with data member total which will store total to subject 1 and 2.Take appropriate member function to initialize the data member and display roll no , individual marks and total marks of a particular student through result class object(Example of multilevel inheritance)

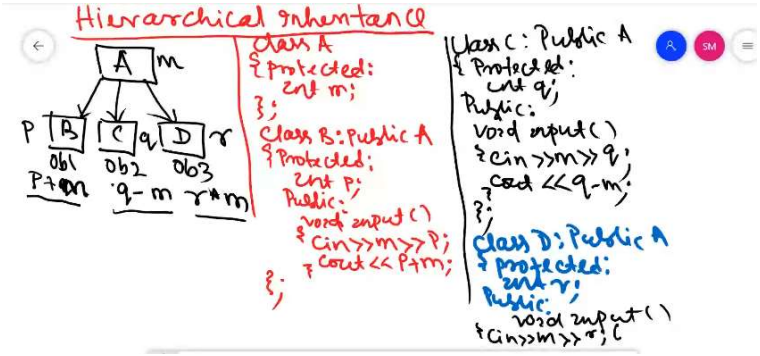
```
#include<iostream>
using namespace std;
class student{
   protected:
   int roll;
};
class test:public student{
  protected:
  int marks1,marks2;
};
class result:public test{
   private:
   int total;
   public:
   void display(){
      cout<<"Enter roll: ";
      cin>>roll;
      cout<<"Enter marks for subject 1 and subject 2: ";
      cin>>marks1>>marks2;
      cout<<"Roll number: "<<roll<<endl;
      cout<<"Marks of subject 1: "<<marks1<<endl;
      cout<<"Marks of subject 2: "<<marks2<<endl;
      total = marks1 + marks2;
      cout<<"Total marks: "<<total<<endl;
   }
};
int main(){
   result ob;
   ob.display();
   return 0;
}
```

xiii. Multiple Inheritance



WAP to create a class ‘student’ having name, roll no, age and section as its data members. Create another class ‘marks’ that stores three subject marks. Derive a class ‘result’ from student and marks class, that stores total and average marks. Display all the details of a student by calling the necessary member functions.

```
#include<iostream>
usingnamespacestd;
classstudent{
protected:
introll;
stringname;
intage;
charsec;
};
classmarks{
protected:
intmarks1,marks2,marks3;
};
classresult:publicmarks,publicstudent{
private:
inttotal;
public:
voiddisplay(){
cout<<"Entername:";
cin>>name;
cout<<"Enterroll:";
cin>>roll;
cout<<"Enterage:";
cin>>age;
cout<<"Entersec:";
cin>>sec;
cout<<"Entermarksforsubject1andsubject2andsubject3:";
cin>>marks1>>marks2>>marks3;
cout<<"Name:"<<name<<endl;
cout<<"Age:"<<age<<endl;
cout<<"section"<<sec<<endl;
cout<<"Roll number:"<<roll<<endl;
cout<<"Marksofsubject1:"<<marks1<<endl;
cout<<"Marksofsubject2:"<<marks2<<endl;
cout<<"Marksofsubject3:"<<marks3<<endl;
total=marks1+marks2+marks3;
cout<<"Total marks:"<<total<<endl;
}
};
Int main(){
Result ob;
ob.display();
Return 0;
}
```

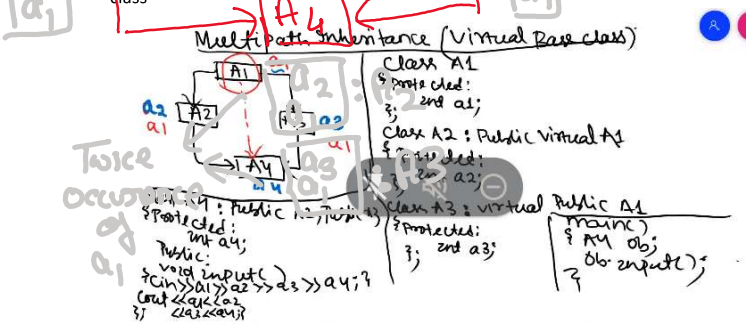
15. Hierarchical Inheritance
    i. 

    ii.
16. Multi Path Inheritance
    i. Virtual Base Class

AS per this diagram all public and protected member of class A1 are inherited to class A4 twice. First via class A2 then again via class A3. This means class A4 can have duplicate sets of member inherited from class A1.
This introduces ambiguity and should be avoided

The duplication of inherited members due to this multiple path can be avoided by making common base class as virtual base class

When a class is made virtual base class cpp see to that only one copy of that class is inherited, regardless of how many inheritance path exists between the virtual base class and the derived class



Nested Class

```
class A
{
	class B
	{
		class C
		{
		};
	};
};

main()
{
A ob1;
A:: B ob2;
A::B::C ob3;
}
```

Ambiguity Resolution in Inheritance

clay, ア (レ ク 0 し 50 ・ ノ 1 ・ と 0 イ

Ambiguity Resolution in Multiple Inheritance

c · — 00 —

Q)

- As long as no base class constructor takes any arguments, the derived class need not have a constructor function.
- If any base class contains a constructor with one or more arguments, then it is mandatory for the derived class to have a constructor and pass the arguments to the base class constructor.
- While applying inheritance, we usually create object using derived class. Thus it makes sense for the derived class to pass arguments to the base class constructor.
- When both the base and derived class contain constructors, the base class constructor is executed first and then the derived class constructor.
- In case of multiple inheritance, the base classes are constructed in the order in which they appear in the declaration of the derived class.

イ ← ~ ア ノ ・ ~ ア ノ ・

The general form of defining derived class constructor if the base class contain parameterized constructor

يالا زه أو ليه لنم«الإسون - وبي ٤٠ ٩٠ : اسمت (83)) مده؟ (2كد


we.