



Inheritance

Chapter 5

Object Oriented Programming

By DSBaral





Introduction

- Inheritance is a process of organizing information in a hierarchical form.
- Through inheritance we can create new class, called derived class, from existing class called base class.
- The derived class inherits all the features of the base class and can add new extra features.
- Inheritance allows new classes to be built from existing and less specialized class instead of writing from the scratch.
- The concept of derived class provides relationships between classes forming hierarchy of classes showing commonality between them.

OOP, Inheritance by DSBaral



2



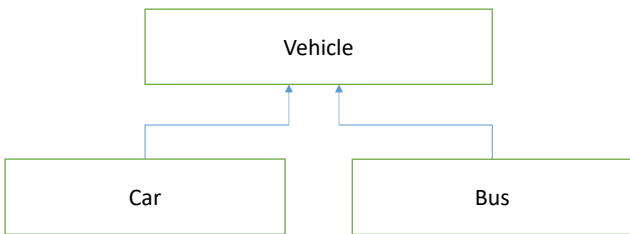
Introduction

- For example the concept of a car and bus are related because they belong to same class vehicle.
- So, it is better to define class car and class bus to have class vehicle in common.
- We can define car and bus classes without creating the vehicle class but this lacks the important feature of OOP.
- So we can create car and bus classes inheriting features from vehicle class which provides the code reusability feature of OOP.
- The great advantage of inheritance is reusability, which ensures ease of distributing class libraries.

OOP, Inheritance by DSBaral 3





Introduction



```
graph BT; Car --> Vehicle; Bus --> Vehicle;
```

Figure: Car and Bus inherited from Vehicle

OOP, Inheritance by DSBaral 4





Base Class and Derived Class

- Inheritance is a technique of building new classes from the existing classes.
- In the process of inheritance, the existing classes that are used to derive new classes are called *base classes* and the new classes derived from existing classes are called *derived classes*.
- When a class is derived from a base class, the derived class inherits all the characteristics of base class and can add new features as refinements and improvements.
- A base class is also called ancestor, parent or super class and derived class is also called as descendent, child or subclass but base class and derived class are preferred terms.

OOP, Inheritance by DSBaral

5



Base Class and Derived Class (Contd...)

Base class

Feature A

Feature B

Feature C

Derived class

Feature D

Feature A

Feature B

Feature C

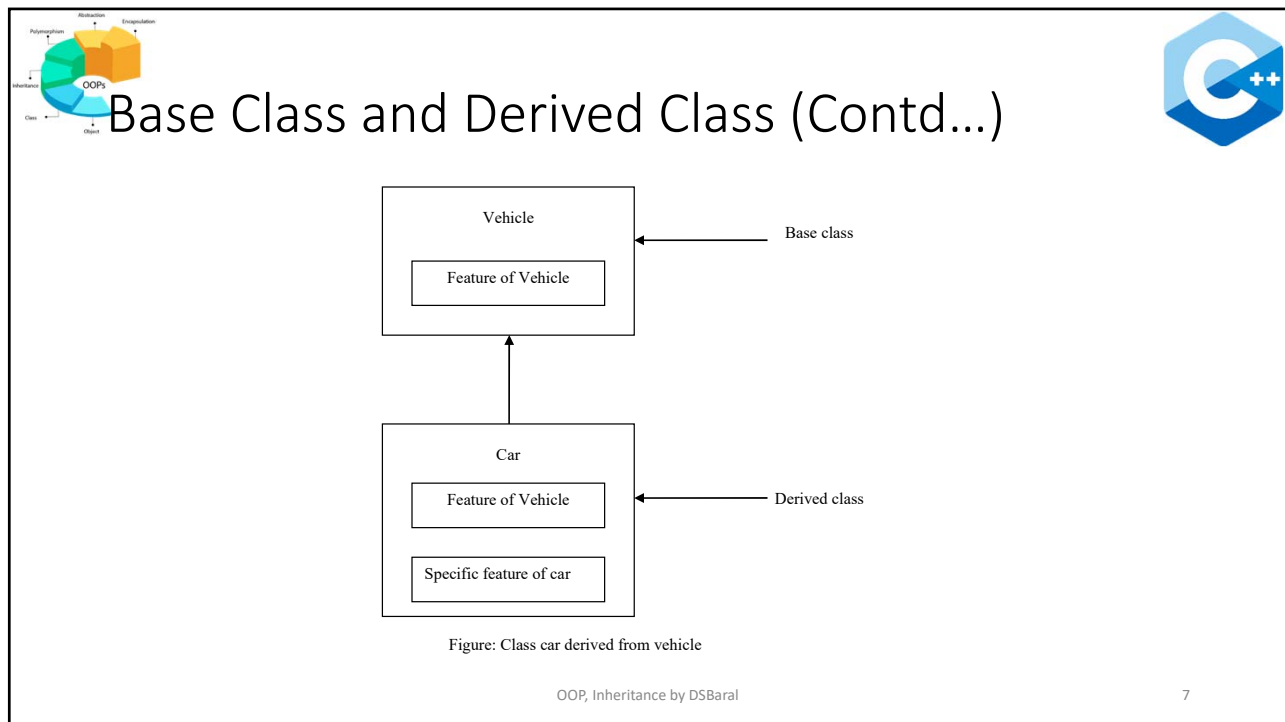
Defined in derived class

Inherited from base class .

Figure: Relation between base and derived classes

OOP, Inheritance by DSBaral

6

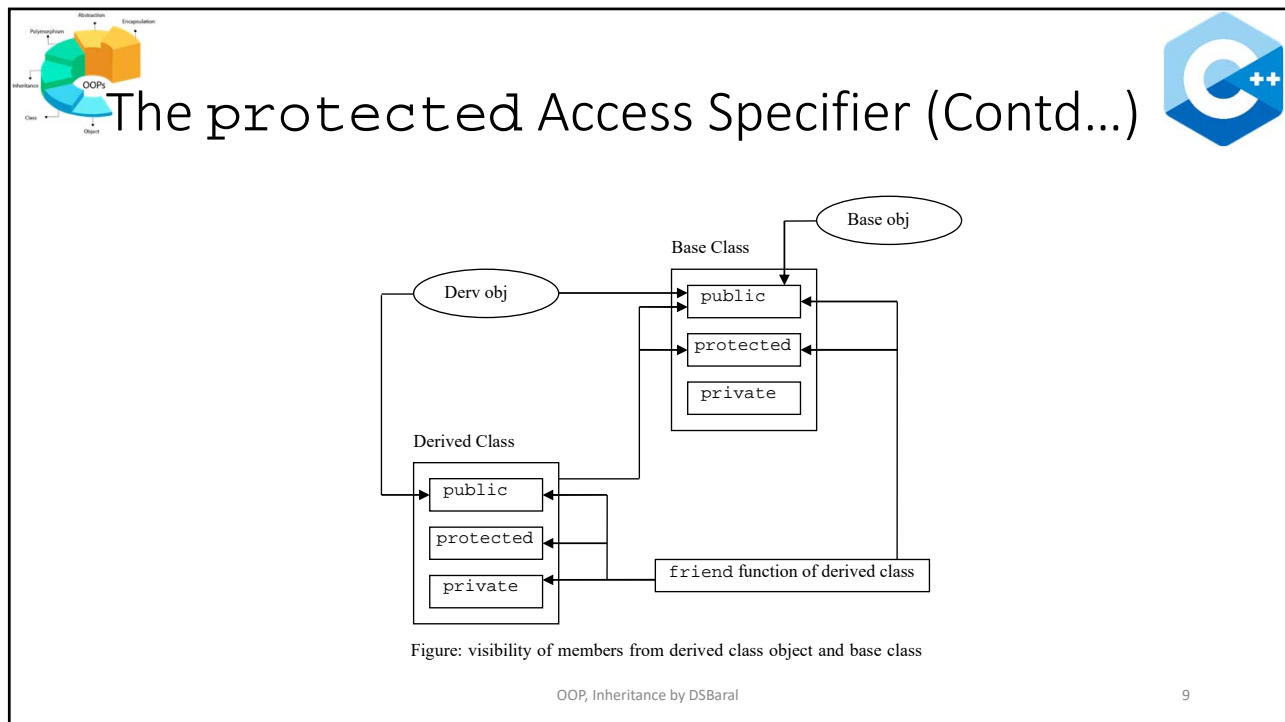


The protected Access Specifier

- The `protected` access specifier causes a member to be visible by the member function and friend function of a class where it is declared but is not visible outside the class similar to `private` access specifier.
- The `protected` members are visible to derived class members and friend of the classes derived from the base class.
- However, `private` members in the base class are not visible in the derived class.
- Declaring data member `protected` is considered less secure leaving data open to the derived class.

OOP, Inheritance by DSBaral



8



The protected Access Specifier (Contd...)

- Following points are to be considered when making members private, public or protected.
 - A private member is accessible only to member/friend functions of the class in which they are declared.
 - A private member of the base class can be accessed in the derived class through the public member functions of the base class only.
 - A protected member is accessible to members of its own class and any of the members in a derived class or friend of derived class.
 - A public member is accessible to members of its own class, member of derived class and even outside the class.

OOP, Inheritance by DSBaral





Derived Class Declaration

- During the derived class declaration we specify the access specifier to the base class to specify access privilege to its members.
- The syntax for the declaration of derived class is as follows:

```
class derived_class:[access_to_base]base_class
{
    // members of derived class
};
```
- The `access_to_base` is optional and can be `public`, `protected` or `private`, and if not specified it will be `private` in class and `public` in struct declaration
 - It specifies the access specifier for the access to base class members from derived class.

OOP, Inheritance by DSBaral 11





Derived Class Declaration (Contd...)

- Following are some possible derive class declarations:
 1. public derivation

```
class derived: public base
{
    //members
};
```
 2. protected derivation

```
class derived: protected base
{
    //members
}
```

OOP, Inheritance by DSBaral 12





Derived Class Declaration (Contd...)

3. private derivation

```
class derived: private base
{
    //members
};
```
4. private derivation by default

```
class derived: base
{
    //members
};
```

OOP, Inheritance by DSBaral 13





Derived Class Declaration (Contd...)

5. public derivation by default

```
struct derived: base
{
    //members
};
```

- In any of the above inheritance,
 - Private data members of the base class cannot be accessed by any of the derived classes.
 - Protected members of the base class are visible in derived class and not from outside except friend functions and classes.
 - Public members are visible in derived class as well as from outside of the classes.

OOP, Inheritance by DSBaral 14






Derived Class Declaration (Contd...)

Base class member access specifiers	Visibility of inherited base class member in derived class		
	public derivation	protected derivation	private derivation
private	invisible	invisible	invisible
protected	protected	protected	private
public	public	protected	private

OOP, Inheritance by DSBaral

15

Derived Class Declaration (Contd...)

- Following example illustrates the concept of inheritance



```

class person{
private:
    char name[25];
    int age;
public:
    void getdata(){
        cout<<"\n Enter Name: "; cin>>name;
        cout<<" Enter Age: ";  cin>>age;
    }
    void showdata(){
        cout<<"\n Name: "<<name;
        cout<<"\n Age:"<<age;
    }
};

```

OOP, Inheritance by DSBaral

16





Derived Class Declaration (Contd...)

```
class student:public person{
private:
    int rollno, score;
public:
    void getsdata(){
        getdata();
        cout<<"\n Enter Roll No.:";cin>>rollno;
        cout<<" Enter Score: "; cin>>score;
    }
    void showsdata(){
        showdata();
        cout<<"\n Roll No: "<<rollno;
        cout<<"\n Score: "<<score;
    }
};
```

OOP, Inheritance by DSBaral

17




Derived Class Declaration (Contd...)

```
int main()
{
    student st;
    cout<<"Enter data of a student"<<endl;
    //st.getdata(); //not a good way
    st.getsdata();
    cout<<"\n Data of student is"<<endl;
    //st.showsdata();
    st.showsdata();
    return 0;
}
```

OOP, Inheritance by DSBaral


18



Member Overriding

- When defining a derived class, the data members and member functions can have the same name as that of the base class.
- Unlike overloading, the functions in the derived class can have same function name and same number and type of arguments.
- The process of creating members in the derived class with the same name as that of the visible members of the base class is called overriding.
- After overriding, when the members are accessed with the overridden names in derived class or through the object of the derived class the derived class members are accessed.
- Even after overriding, we can even access overridden base class members.

OOP, Inheritance by DSBaral 19





Member Overriding (Contd...)

- Following example illustrates concept of inheritance and data member and function member overriding.

```
class base{
    protected:
        int num;
    public:
        void readdata(){
            cout<<"Enter number in base: ";cin>>num;
        }
        void showdata(){
            cout<<"Number in base= "<<num<<endl;
        }
};
```

OOP, Inheritance by DSBaral 20


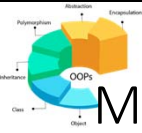


Member Overriding (Contd...)

```
class derived : public base{
private:
    int num;
public:
    void readdata()
    {
        cout<<"Enter number in derived: ";
        cin>>num;
    }
    void showdata()
    {
        cout<<"Number in derived class="<<num<<endl;
    }
};
```

OOP, Inheritance by DSBaral

21



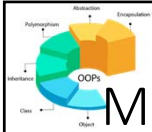
Member Overriding (Contd...)

```
int main()
{
    derived d1;
    d1.readdata();
    d1.showdata();
    return 0;
}
```

- Here the derived class object d1 calls the derived class readdata () and showdata () functions
- After overriding, the base class members can be called through derived class functions or through objects.

OOP, Inheritance by DSBaral

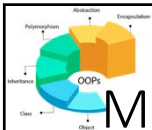
22



Member Overriding (Contd...)

- When the base class function is overridden, the base class and derived class functionality is achieved by calling base class and derived class functions separately as.



```
int main()
{
    derived d1;
    d1.base::readdata();//read base::num
    d1.readdata();//read derived class num
    d1.base::showdata();//displays base::num
    d1.derived::showdata();//same as d1.showdata()
    return 0;
}
```



Member Overriding (Contd...)

- But this method is tedious, cumbersome and error prone so the base class functionality can be embedded in derived class function as



```
class derived:public base{
    //.....
public:
    void readdata(){
        base::readdata();
        cout<<"Enter number in derived: ";cin>>num;
    }
    void showdata(){
        base::showdata();
        cout<<"Number in derived class="<<num<<endl;
        cout<<"Base member from derived="<<base::num<<endl;
    }
};
```



Forms of Inheritance

- A base class for any derived class can also be a derived class of other classes.
- The level of inheritance is the length of the path from the top base class to the bottom derived class.
- There are following form of inheritance based on the levels of inheritance and interrelation among the classes involved in the inheritance process.
 - a) Single
 - b) Multiple
 - c) Multilevel
 - d) Hierarchical
 - e) Hybrid
 - f) Multipath

OOP, Inheritance by DSBaral 25





Single Inheritance

- When a class is derived from only one base class then such derivation is called single inheritance.
- In a single inheritance, base class and derived class exhibits one to one relationship.
- The syntax of single inheritance is given below:

```
class derived:[access_to_base]base
{
    //.....
}
```

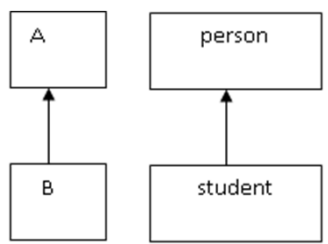
OOP, Inheritance by DSBaral 26

Single Inheritance (Contd...)



- The usage of single inheritance along with the figure is as follows:


```
class A
{
    //.....
};
//B is derived from class A.
class B : public A
{
    //.....
};
```
- The student class derived from person discussed earlier is the example of single inheritance



OOP, Inheritance by DSBaral

27



Multiple Inheritance

- When a class is derived from two or more base classes, then such type of inheritance is called multiple inheritance.
- Multiple inheritance allows the combination of more than one existing classes as base class for the creation of new derived class.
- Multiple inheritance has the following syntax


```
class derived:[access_to_base]base1,[access_to_base]base2
{
    //member of derived class
}
```

OOP, Inheritance by DSBaral

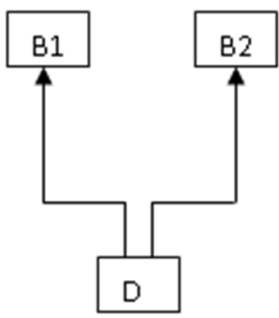
28



Multiple Inheritance (Contd...)

- Following declaration along with the figure illustrates the concept of multiple inheritance.



```
class B1{  
    //.....  
};  
  
class B2{  
    //.....  
};  
class D:public B1,public B2  
{  
    //.....  
};
```



```
graph BT; D --> B1; D --> B2;
```

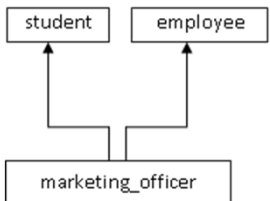
OOP, Inheritance by DSBaral

29



Multiple Inheritance (Contd...)

- Multiple inheritance can be illustrated by the following example





```
graph BT; marketing_officer --> student; marketing_officer --> employee;
```

```
class student  
{  
private:  
    char name[25];  
    int studID;
```

OOP, Inheritance by DSBaral

30

Multiple Inheritance (Contd...)



```

public:
    void getdata(){
        cout<<"\n Enter Name:"; cin>>name;
        cout<<" Enter Student ID: "; cin>>studID;
    }
    void showdata(){
        cout<<"\n Name: "<<name;
        cout<<"\n Student ID:"<<studID;
    }
};
class employee {
private:
    char org_name[25];
    int empID;

```

OOP, Inheritance by DSBaral

31

Multiple Inheritance (Contd...)



```

public:
    void getdata(){
        cout<<"Enter Name of Organization: "; cin>>org_name;
        cout<<"\nEnter employeeID: "; cin>>empID;
    }
    void showdata(){
        cout<<"\nName of Organization: "<<org_name;
        cout<<"\nEmployeeID: "<<empID;
    }
};
class marketing_officer:public student,public employee
{
private:
    int working_hour;

```

OOP, Inheritance by DSBaral

32


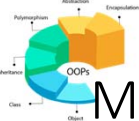


Multiple Inheritance (Contd...)

```
public:
    void getdata(){
        student::getdata();
        employee::getdata();
        cout<<"Enter working hours: ";
        cin>>working_hour;
    }
    void showdata(){
        student::showdata();
        employee::showdata();
        cout<<"\n Working hour: "<<working_hour;
    }
};
```

OOP, Inheritance by DSBaral

33





Multiple Inheritance (Contd...)

```
int main()
{
    marketing_officer moff;
    cout<<"Enter data of marketing officer"<<endl;
    moff.getdata();
    cout<<"\n Data of marketing officer"<<endl;
    moff.showdata();
    return 0;
}
```

- In this example if we have to do something more with base class members in derived class we could have made the base class members as protected.

OOP, Inheritance by DSBaral



34

Ambiguity in Member Access in Multiple Inheritance

- In multiple inheritance if more than one base classes have member functions with the same name then problem of ambiguity arises.
 - The ambiguity arises when accessing those member functions from the derived class or the object of the derived class.
- Ambiguity arises because the compiler cannot determine which member function from which base class to call.
- Ambiguity also arises when derived class function or object of derived class refers a data member whose name is same in multiple base classes.
- The ambiguity can be eliminated by referring to the members with base class name and scope resolution operator before member name.

OOP, Inheritance by DSBaral 35



Ambiguity in Member Access in Multiple Inheritance

- The following program illustrates this case of ambiguity

```

class BaseOne{
public:
    void display() {cout<<"I am from BaseOne"<<endl; }
};
class BaseTwo{
public:
    void display() {cout<<"I am from BaseTwo " <<endl; }
};
class Derived : public BaseOne, public BaseTwo{
public:
    void drvdisplay(){
        display();//error:ambiguous, which display to call
    }
};
  
```

OOP, Inheritance by DSBaral 36



Ambiguity in Member Access in Multiple Inheritance (Contd...)

```

        BaseOne::display();//ok: base class name is specified
        BaseTwo::display();//ok: base class name is specified
    }
};
int main()
{
    Derived Der;
    Der.display() ; //error:ambiguous,which display to call
    Der.BaseOne::display();
        //display()of BaseOne class is called
    Der.BaseTwo::display();
        //display()of BaseTwo class is called
}

```

OOP, Inheritance by DSBaral 37

Ambiguity in Member Access in Multiple Inheritance (Contd...)

- The other solution is to define display in derived class as


```

class Derived : public BaseOne , public BaseTwo{
public:
    void display(){
        BaseOne::display();//ok: base class specified
        BaseTwo::display();//ok: base class specified
        //Derived display part ...
    }
};



```
- Then


```

Derived der;
der.display();//ok: after defining display in derived

```

OOP, Inheritance by DSBaral 38

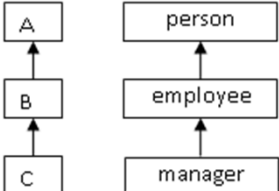



Multilevel Inheritance

- A new class that is derived from a base class can again serve as a base for further derivation.
- The derivation of a class from another derived class is called multilevel inheritance.
- The multilevel inheritance is declared as follows and the corresponding figure illustrates the concept.



```

class A{ //...
};
class B:public A{ //...
};
class C:public B{
    //...
};
  
```



OOP, Inheritance by DSBaral

39





Hierarchical Inheritance

- When different classes are derived from single base class, such type of inheritance is called hierarchical inheritance.
- The new classes can still be base class for other new classes.
- This model provides a hierarchy of classes.
- The following declaration and figure illustrates this concept


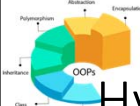
```

class B:public A{ //...
};
class C:public A{ //...
};
class D:public A{ //...
};
  
```



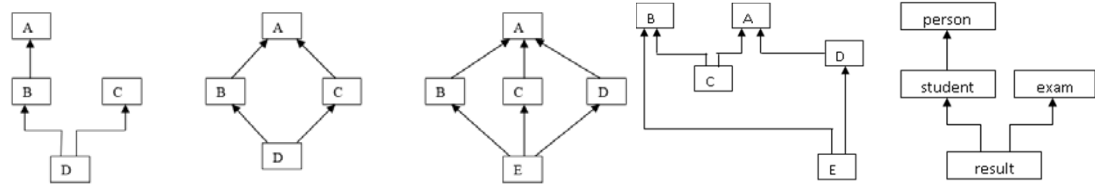
OOP, Inheritance by DSBaral

40


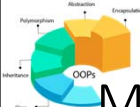
Hybrid Inheritance

- When we mix more than one form of inheritance we call it hybrid inheritance.
- Following are some possible hybrid Inheritances.



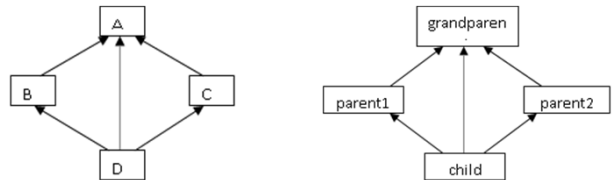
OOP, Inheritance by DSBaral

41

Multipath Inheritance and Virtual Base Class



- Derivation of a class from other derived classes, which are derived from same base class, is called multipath inheritance.
- Multipath inheritance involves other inheritance like multiple, multi level, and hierarchical.



- Features of the same base class A are inherited to its derived classes B and C are inherited to the final derived class D from two paths.

OOP, Inheritance by DSBaral

42





Multipath Inheritance and Virtual Base Class (Contd...)

- Let's see the following example

```
class grandparent
{
private:
    int data;
public:
    void setdata(int n){data=n;}
    void showdata(){cout<<data;}
};
class parent1:public grandparent
{
    //.....
};
```



OOP, Inheritance by DSBaral 43



Multipath Inheritance and Virtual Base Class (Contd...)

```
class parent2:public grandparent
{
    //.....
};
class child:public parent1,public parent2
{
public:
    void showchdata()
    {
        showdata();//ambiguous which showdata()
    }
};
```

OOP, Inheritance by DSBaral 44



Multipath Inheritance and Virtual Base Class (Contd...)

- This ambiguity can be resolved by indicating that a class grandparent is a virtual base class while creating class parent1 and parent2 as

```

class parent1:virtual public grandparent {
    //.....
};
class parent2:virtual public grandparent{
    //.....
};
class child:public parent1,public parent2{
public:
    void showchdata(){
        showdata(); //ok: only one copy of grandparent
    }
};
  
```

OOP, Inheritance by DSBaral 45



Multipath Inheritance and Virtual Base Class (Contd...)

- Even without making virtual base class the ambiguity can be resolved as

```

class child:public parent1,public parent2
{
public:
    void showchdata()
    {
        grandparent::showdata();
        //still ambiguous because parent1 and parent2 both
        //have gradparent::showdata();
        showdata();//ambiguous as said earlier
        parent1::showdata();//ok not ambiguous
        parent2::showdata();//ok not ambiguous
    }
};
  
```

OOP, Inheritance by DSBaral 46





Multipath Inheritance and Virtual Base Class (Contd...)

- The `showchdata ()` function could also be defined to have the same name of `grandparent` as

```
class child:public parent1,public parent2
{
public:
    void showdata()
    {
        grandparent::showdata();//error
        parent1::showdata();//ok not ambiguous
        parent2::showdata();//ok not ambiguous
    }
};
```

OOP, Inheritance by DSBaral 47





Multipath Inheritance and Virtual Base Class (Contd...)

- The members can be accessed through the object as

```
child chl;
chl.showdata();
//works well: overrides function of child is called
chl.parent1::showdata();//ok
chl.parent2::showdata();//ok
chl.grandparent::showdata();
//error: ambiguous;there are two version
// of grandparent::showdata
```



OOP, Inheritance by DSBaral 48



Constructors in Derived Class

- In inheritance, when object of derive class is created following things happen.
 - 1) Base class constructors are invoked first and derived class constructors are invoked next
 - 2) If we do not explicitly specify base class constructors, the default constructors from base class is invoked
 - 3) To invoke parameterized constructors in base class they must be specified explicitly in derived class constructor.
- The various cases of constructors in base and derived class are discussed below



OOP, Inheritance by DSBaral 49



Constructors in Derived Class (Contd...)

```
class base
{
public:
    base()
    {
        cout<<"default constructor from base class"<<endl;
    }
    base(int data)
    {
        cout<<"parameterized const from base class"<<endl;
    }
};
```

OOP, Inheritance by DSBaral 50

Constructors in Derived Class (Contd...)



```

class derived:public base
{
public:
    derived()
    {
        cout<<"default constructor from derived class"<<endl;
    }
    derived(int data)
    {
        cout<<"parameterized constructor from derived class"<<endl;
    }
};
int main()
{
    derived d1,d2(5);
}

```

OOP, Inheritance by DSBaral

51

Constructors in Derived Class (Contd...)

- In both of the object creation the default constructor of base is called.
- To invoke the base class parameterized constructor it is to be specified explicitly from the derived class as



```

class derived:public base
{
public:
    //...
    derived(int data):base(data)
    {
        cout<<"parameterized constructor from derived class;
    }
};

```

OOP, Inheritance by DSBaral

52

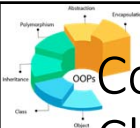



Constructor for Multiple Inherited Derived Classes

- When derived class are created by inheriting multiple base class then the constructors from all the base classes are invoked first and the derived class constructor is called next.
- The order of constructor invocation depends on the order of how base class is inherited. For example

```
class D:public B1,public B2
{
    //.....
};
```
- In this case B1 is inherited first so the constructor of the class B1 is called first and the constructor of the class B2 is called next.

OOP, Inheritance by DSBaral 53





Constructor for Multiple Inherited Derived Classes (Contd...)

- Let's see the following example for member initialization

```
class B1
{
private:
    int a;
public:
    B1(){}
    B1(int n){a=n};
};
```

OOP, Inheritance by DSBaral 54





Constructor for Multiple Inherited Derived Classes (Contd...)

```
class B2
{
private:
    int b;
public:
    B2(){}
    B2(int n){b=n;}
};
```

OOP, Inheritance by DSBaral

55





Constructor for Multiple Inherited Derived Classes (Contd...)

```
class D:public B1, public B2
{
private:
    int d;
public:
    D(){}
    D(int n1,int n2,int n3):B2(n2),B1(n1),d(n3){}
};
```

- Here the order of constructor call is B1, B2 and D irrespective of the order in initializer list.

OOP, Inheritance by DSBaral


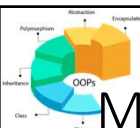
56

Destructor in Derived class

- The order of the constructor invocation when derived class is created is from base class to derived class.
- In multiple inheritance the order of base class in the inheritance specifies the constructor invocation order.
- In multilevel inheritance, the constructor of the root base class is called at first and the constructor of the lastly derived class is called sequentially at last.
- Unlike constructor call order, the order of destructor invocation is just in reverse order of constructor invocation order.
- The destructor of the class whose constructor is called at last is called first and the destructor of the class whose constructor is called at first is called at last.

OOP, Inheritance by DSBaral 57

Member Initialization Order

- Let's see a case in what order the class members are initialized.

```
class B{
private:
    int x;
    int y;
public:
    B(int a,int b):x(a),y(b){}
};
```

- The other way of constructor definition can be

```
B(int a,int b):x(a),y(a+b){} //ok
B(int a,int b):x(a),y(x+b){} //ok
B(int a,int b):y(a),x(y+b){}
//wrong: x is initialized first
```

OOP, Inheritance by DSBaral 58