Object Oriented Programming in C++

Nepal College of Information Technology

Course Instructor: Er. Rabina Chaudhary

Chapter 4: Inheritance

Function Overriding:

- Two or more functions having same name and same signature (number, type and sequence of parameters) but one defined in base class and other defined in derived class
- When the function is called with the help of object of derived class, by default derived class gives priority to the function defined within it.
- So base class function is overridden by derived class version

Example program:

```
#include<iostream.h>
#include<conio.h>
class Base
{
        public :
                void display()
                {
                        cout<<"This is base "<<endl;
                }
};

class Derived: public Base
{
        public :
                void display()
                 {
                        cout<<"This is derived "<<endl;
                }
};

void main()
{
        Base bobj;
        cout<<"Calling from Base class's object"<<endl;
        bobj.display();
        Derived dobj;
        cout<<endl<<endl<<"Calling from Derived class's object"<<endl;
        dobj.display();
        dobj.display();
        getch();
}
```

Function Overriding :

- We use base class name and scope resolution operator to call base class version using object of derived class

Example program:

```
#include<iostream.h>
#include<conio.h>
class Base
{
        public :
                void display()
                {
                        cout<<"This is base "<<endl;
                }
};

class Derived: public Base
{
        public :
                void display()
                 {
                        cout<<"This is derived "<<endl;
                }
};

void main()
{
        Base bobj;
        cout<<"Calling from Base class's object"<<endl;
        bobj.display();
        Derived dobj;
        cout<<endl<<endl<<"Calling from Derived class's object"<<endl;
        dobj.Base::display();
        dobj.display();
        getch();
}
```

Function Overriding:

- Two or more functions having same name and same signature (number, type and sequence of parameters) but one defined in base class and other defined in derived class
- In function overriding, the compiler doesn’t know about the correct form of function to be called in compile time. The correct form of function is selected to call on the basis of content of calling object at runtime or while the program is running.
- The code associated with the function call is not known until program execution, hence called late binding/ dynamic binding/ dynamic linkage.

Virtual Function:

- In C++, function overriding is achieved using virtual function.
- A virtual function is a member function which is declared within a base class and overridden by derived class
- When we use same function name with same signature in base class and derived class, the function in base is declared as virtual function
- When the class containing virtual function is inherited, the derived class redefines virtual function to perform task respective to derived class

Virtual Function:

- Base class pointer can point object of same class and also point object of its derived classes
- The correct version of function is called on the basis of type of object pointed by base class pointer at particular time
- By making the base class pointer point to different class's object, we can execute different version of virtual function
- If the derived class doesn't redefine virtual function, the derived class inherits its immediate base class virtual function definition
- Base class pointer can point to object of any of its descendant class

Without implementing Virtual Function : Compile and run this program

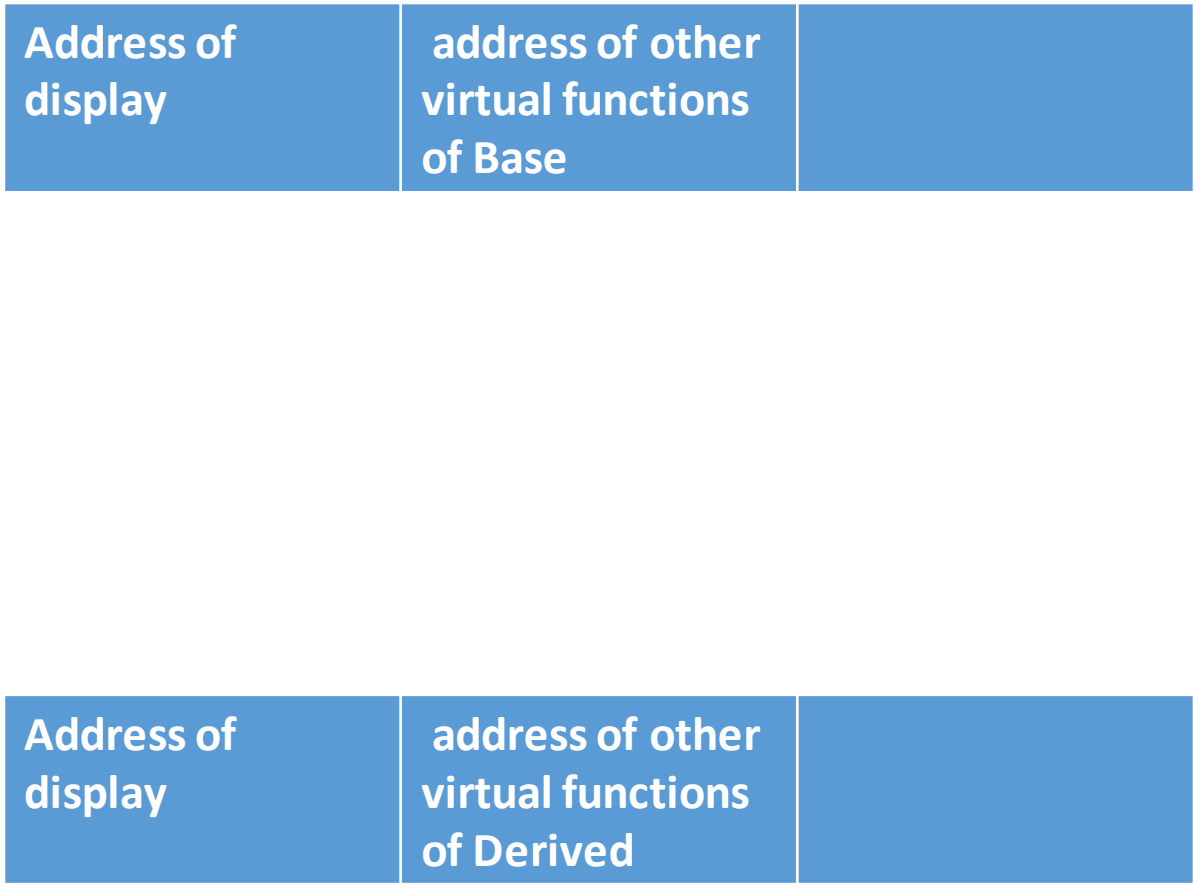
```
#include<iostream.h>
#include<conio.h>

class Base
{
        public :
        void display()
        {
        cout<<"This is base "<<endl;
        }
};

class Derived: public Base
{
        public :
        void display()
        {
        cout<<"This is derived "<<endl;
        }
};

void main()
{
        Base *bptr, bobj;
        bptr=&bobj;
        cout<<"Calling from Base class's pointer
        that points to Base's object"<<endl;
        bptr->display();

        Derived dobj;
        bptr=&dobj;
        cout<<endl<<endl<<"Calling from Base class's
        pointer that points to Derived's object"<<endl;
        bptr->display();
        getch();
}
```

implementing Virtual Function : Compile and run this program

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

class Base
{
        public :
        virtual void display()
        {
        cout<<"This is base "<<endl;
        }
};

class Derived: public Base
{
        public :
        void display()
        {
        cout<<"This is derived "<<endl;
        }
};

void main()
{
        Base *bptr, bobj;
        bptr=&bobj;
        cout<<"Calling from Base class's pointer
        that points to Base's object"<<endl;
        bptr->display();

        Derived dobj;
        bptr=&dobj;
        cout<<endl<<endl<<"Calling from Base class's
        pointer that points to Derived's object"<<endl;
        bptr->display();
        getch();
}
```

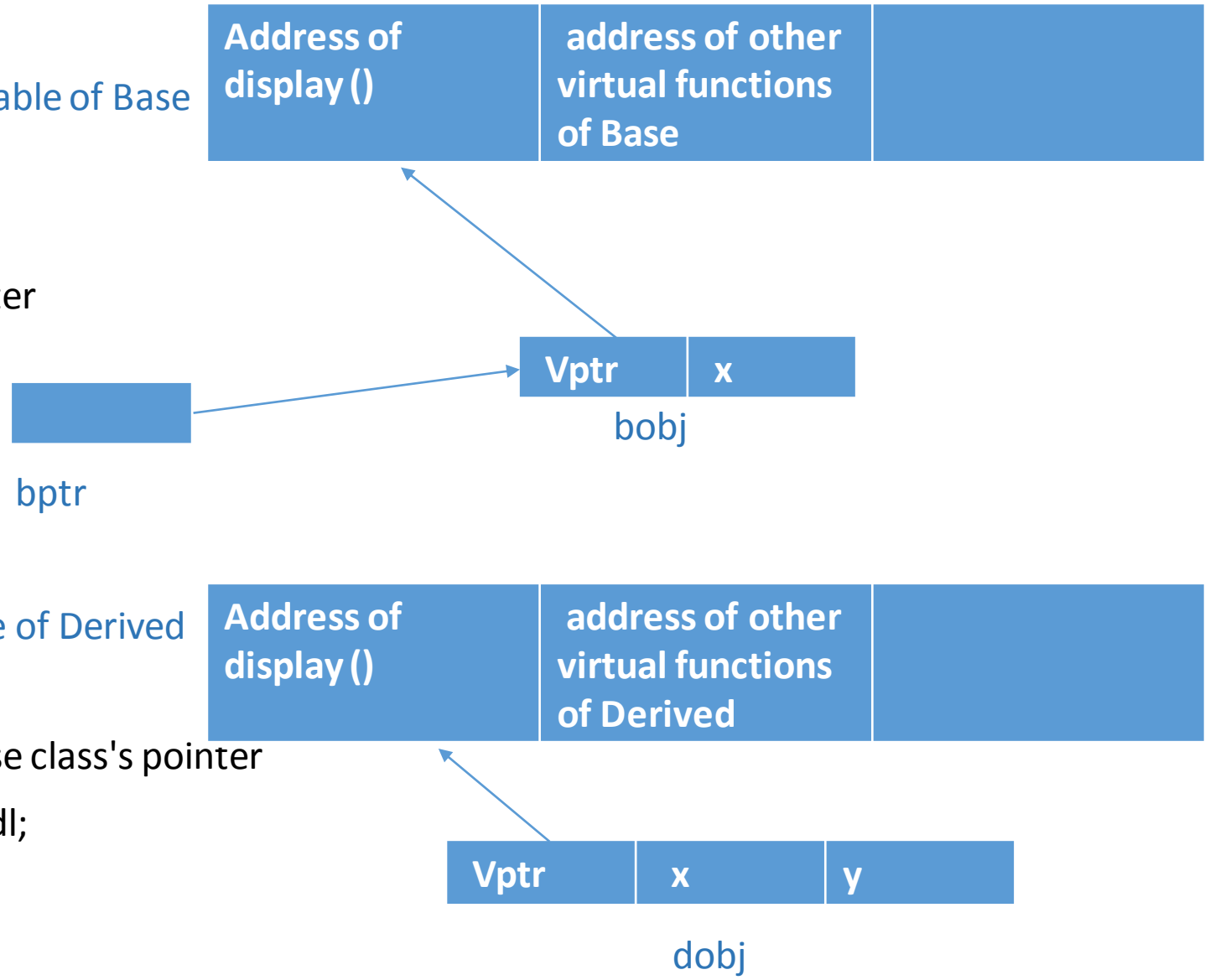
Working of Virtual Function:

```
class Base{
                int x;      Vptr
                public :
                virtual void display()
                {
                cout<<"This is base "<<endl ;
                }
                void testFunction()
                {
                cout<<“This is Test in base “<<endl;
                }
                …
};

class Derived: public Base{
                public :
                int y;
                void display()
                {
                cout<<"This is derived "<<endl;
                }
                void testFunction()
                {
                cout<<“This is Test in base “<<endl;
                }
                …
};
```



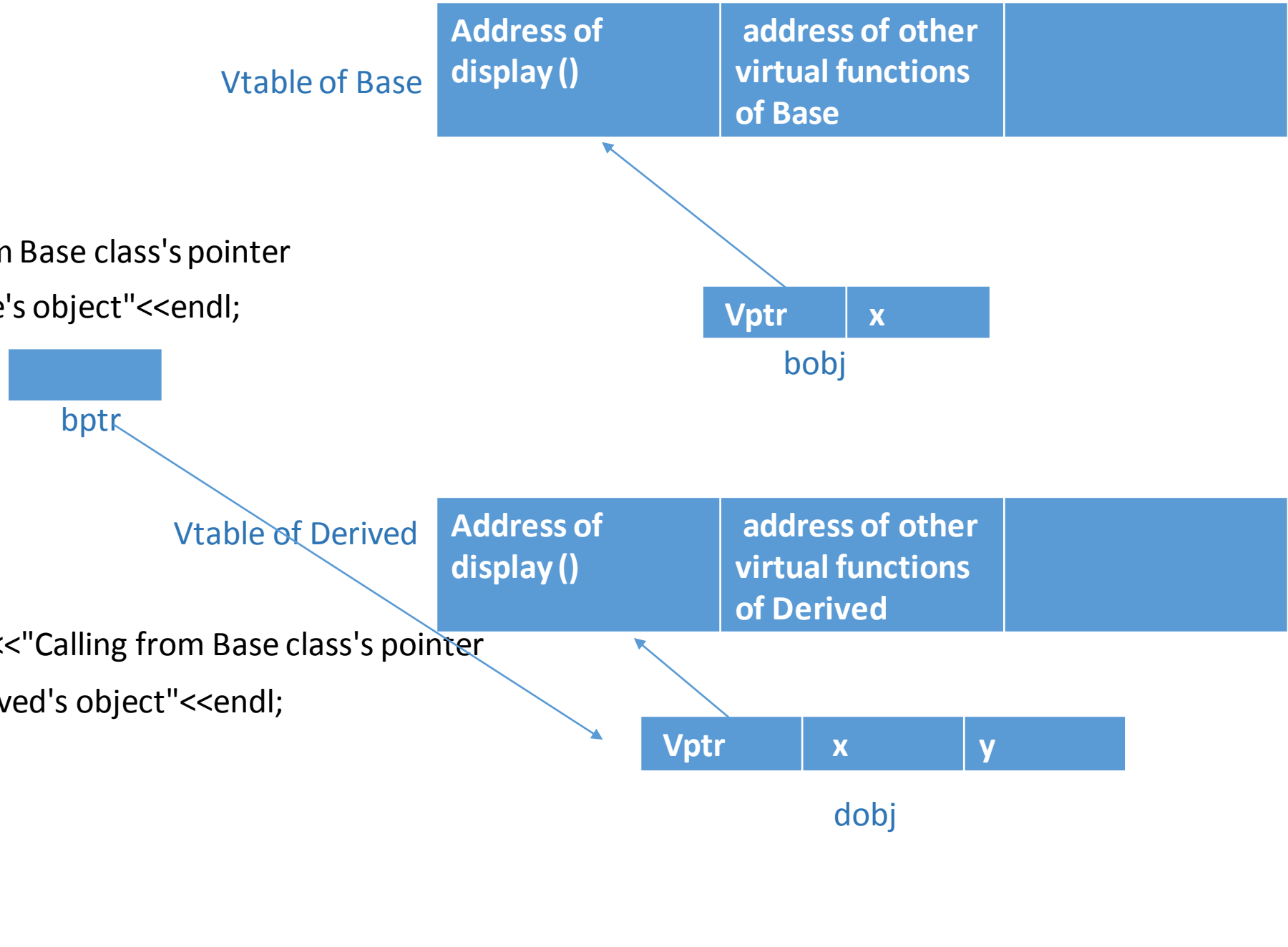
Working of Virtual Function:

```
void main()
{
        Base *bptr, bobj;
         bptr=&bobj;
        cout<<"Calling from Base class's pointer
         that points to Base's object"<<endl;
        bptr->display();

        Derived dobj;
        bptr=&dobj;
        cout<<endl<<endl<<"Calling from Base class's pointer
         that points to Derived's object"<<endl;
        bptr->display();
        getch();
}
```



Working of Virtual Function:

```
void main()
{
        Base *bptr, bobj;
         bptr=&bobj;
        cout<<"Calling from Base class's pointer
         that points to Base's object"<<endl;
        bptr->display();

        Derived dobj;
        bptr=&dobj;
        cout<<endl<<endl<<"Calling from Base class's pointer
         that points to Derived's object"<<endl;
        bptr->display();
        getch();
}
```



Virtual Constructor:

- Constructor cannot be virtual because when constructor of a class is executed, virtual pointer is not defined yet.

Virtual Destructor:

- A virtual destructor is used to free up the memory space allocated by the derived class object while deleting instance of the derived class using base class pointer.

Pure Virtual function:

- Virtual function which has no body is known as pure virtual function
- It is a virtual function for which we do not need to write any function definition, we only need to declare it

Syntax to declare pure virtual function:

```
virtual returnType functionName()=0;
```

- It is used when function doesn’t have any use in the base class but must be implemented by all its derived class

Pure Virtual function:

- Pure virtual is declared in base class and it has no function definition relative to the base class
- Each derived class define the virtual function as per its requirement

Abstract Class:

- It is a class that has at least one pure virtual function
- The classes inheriting the abstract class must provide the definition for the pure virtual function otherwise the subclass would become an abstract class itself
- Abstract class cannot be instantiated, but pointers of abstract class can be created
- Abstract class can have normal functions and variables along with a pure virtual function

Programs:

1. Define a class Person with attributes and behaviors that are common to Student and Teacher, Person class doesn’t have any implementation of the functions, so make it abstract class. Derive class Student and Teacher from class Person and provide the implementation of the inherited functions in each derived class.