TATA

TATA ELXSI

OBJECT ORIENTED PROGRAMMING USING C++
Module 8

Learning & Development Team

Polymorphism

Virtual Functions (Polymorphism)

- A virtual function is a member function that is declared within a base class, and is redefined by a derived class.
- To create a virtual function, precede the function's declaration in the base class with the keyword **virtual**.
- When a class containing a virtual function is inherited, the derived class redefines or overrides the virtual function to fit its own needs.

Virtual Functions

```
class base
 {
   public:
       virtual void vfunc( )
        { cout << “this is base’s vfunc( )\n”; }
};

class derived1: public base
 {
      public:
          void vfunc( )
          {  cout << “this is derived1’s vfunc( )\n”; }
};
```

Virtual Functions

```
class derived2 : public base
{
   public:
   void vfunc( )
   {
    cout << “this is
       derived2’s vfunc( )\n”;
    }
};
```

```
int main( )
{
   base *p, b;
   derived1 d1;
   derived2 d2;
   p = &b;

   p->vfunc( ); //base’s vfunc( )
   p = &d1;

   p->vfunc( ); //d1’s vfunc( )
   p = &d2;

   p->vfunc( ); // d2’s vfunc( )
   return 0;
}
```

Virtual Functions are Hierarchical

- When a function is declared as virtual by a base class, it may be overridden by a derived class. However, the function does not have to be overridden.
- *When a derived class fails to override a virtual function, then, when an object of the derived class accesses that function, the function defined by it's base class is used.*
- Consider this program in which class derived2 does not override vfunc( )

Virtual Functions are Hierarchical

```
#include<iostream>
using namespace std;
class base
{
   public:
    virtual void vfunc( )
    { cout << “this is base’s vfunc( )\n”; }
};

class derived1: public base
{
   public:
   void vfunc( )
   {  cout << “this is derived1’s vfunc( )\n”; }
};
```

Virtual Functions are Hierarchical

```
//derived2 inherits virtual function from derived1
class derived2 : public derived1
 {  // vfunc( ) not overridden by derived2; base's is used  };

int main( )
 {
   base *p, b;
   derived1 d1;
   derived2 d2;
   p = &b;
   p->vfunc( ); //access to base's vfunc( )
   p = &d1;
   p->vfunc( ); // access derived1's vfunc( )
   p = &d2;
   p->vfunc( ); // access base's vfunc( ) since derived2 does not override vfunc()
   return 0;
}
```

Pure Virtual Functions

- When a virtual function is not redefined by the derived class, the version defined in the base class will be used. However, in many situations, there cannot be any meaningful definition of a virtual function within a base class.
- For example, a base class may not be able to define an object sufficiently to allow a base class virtual function to be created.
- Further, in some situations, you will want to ensure that all derived classes compulsorily override a virtual function.

Pure Virtual Functions(Abstract Class)

- To handle these two situations, C++ supports the **Pure Virtual Function.** A pure virtual function is a virtual function that has no definition in the base class.

- To declare a pure virtual function, use this general form:

  virtual type func_name (parameter_list) = 0;

- When a virtual function is declared pure, any derived class **must** provide its own definition of the virtual function.

Pure Virtual Functions(Abstract Class) [Exercise]

Note:

- If we don't override the PVF from base to derived , then that sub class should be an abstract class again
- We cannot create an object or cannot be instantiated for an abstract class.
- We can create a pointer variable for abstract class which can refer to any of its derived class as per the class diagram we design.
- *Do try these above scenarios as a Task…*

Virtual Function Mechanics – The Virtual Table

- C++ implements late binding by setting up a **vtable.** The keyword **virtual** tells the compiler it should not perform early binding.

- Instead, it should automatically install all the mechanisms necessary to perform late binding. The compiler creates a single table called VTABLE for each class that contains virtual functions.

- The compiler places the addresses of the virtual functions for that particular class in the VTABLE. A virtual table is therefore an array of virtual function pointers stored by the compiler as a table called as VTABLE.

Virtual Destructor

- When using dynamic binding all the instances of a class may not be properly disposed off.
- As delete to a pointer to a base class will call the destructor of the base class only.
- But if destructor is declared virtual it will call the inherited class destructor as well, thus properly disposing the class instances.

Friend functions

- In principle, private and protected members of a class cannot be accessed from outside the same class in which they are declared. However, this rule does not affect friends.
- To declare a friend function, include its prototype within the class, preceding it with the keyword friend.
- Derived class does not inherit friend functions.

Friend functions - example

```
#include <iostream>
using namespace std;
class mycl
{
   int a,b;
  public:
   friend int add(mycl x);
   void set_ab(int i, int j);
}
void mycl::set_ab(int i, int j)
{
   a = i, b = j;
}
int add (mycl x)
{
   return x.a + x.b;
}
int main()
{
   mycl cl;
   cl.set_ab(4,5);
   cout << add(cl);
   return 0;
}
```

Friend functions

- The keyword friend is placed only in the function declaration of the friend function and not in the function definition.
- It is possible to declare a function as friend in any number of classes.
- When a class is declared as a friend, the friend class has access to the private data of the class that made this a friend.
- A friend function, even though it is not a member function, would have the rights to access the private members of the class.
- It is possible to declare the friend function as either private or public.
- The function can be invoked without the use of an object. The friend function can have its argument as objects.

Friend Class

- A friend class must be previously declared in an enclosing scope.
- Scopes outside the innermost enclosing namespace scope are not considered.
- A friend function can be explicitly declared just like friend class.

Friend functions

```
class mycl
{
    int x;
    friend void fun(mycl);
};

void fun( mycl o)
{
    o.x = 500;
    cout << o.x << endl;
}
```

```
void fun1(mycl x)
{
    fun (x);
}

int main()
{
    mycl o;
    fun1(o);
}
```

Friend Classes

- A class can be a friend of another class.
- The friend class member functions has access to the private members defined within the class.
- When one class is a friend of another, it only has access to names defined within other class. It does not inherit the other class.

Friend Classes - example

```
#include <iostream>
using namespace std
class mycl
{
   int a,b;
  public:
   friend class myadd;
   void setab(int i,  j);
};
void mycl::setab(int i, int j)
{
   a = i;
   b = j;
}
```

```
class myadd
{
 public:
   int add(mycl x)
   { return x.a + x.b; }
};

int main()
{
   mycl cl; myadd ad;
   cl.setab(4,5);
   cout << ad.add(cl);
   return 0;
}
```

Name-Space

- Namespaces allow to group entities like classes, objects and functions under a name. This way the global scope can be divided in "sub-scopes", each one with its own name.

- The format of namespaces is:

```
namespace identifier
{
        entities
}
```

Are you ready to solve…

1. A friend function cannot access public members of another class, It can access only private members.
   a. True
   b. False

   Ans: **b. False**

2. C++ implements late binding by setting up a

   a. PeriodTable b. Stack c. FunctionTable d. VTable

   Ans: **d. VTable**

End of Module 8

Disclaimer

- Some examples and concepts have been sourced from the below links and are open source material
  - **http://cppreference.com**
  - **www.cplusplus.com**

- References:
  - *C++: The Complete Reference* - 4th Edition by Herbert Schildt, Tata McGraw-Hill publications.
  - *The C++ Programming Language*- by Bjarne Stroustrup.
  - *Practical C++ Programming*- by Steve Oualline, O'Reilly publications.

Learning & Development Team

ITPB Road Whitefield
Bangalore 560 048 India
Tel +91 80 2297 9123
Fax +91 80 2841 1474
e-mail info@tataelxsi.com

www.tataelxsi.com