Unit 8 Polymorphism and Virtual Functions

Structure:

8.1 Introduction

In the previous unit you have learnt about inheritance and its types. You have also studied in detail the concept of multiple inheritance. In this unit you are going to study polymorphism and virtual functions. Polymorphism refers to the ability to call different functions by using only one type of function call. The special type of functions which can be re-defined in derived classes are known as virtual functions. Virtual functions are special member functions of a class which may be re-defined in the derived classes. It is used to give specific meaning to the base class member function with respect to the derived class. Virtual functions can be thought of as a function name reserved in the base class which may be re-defined in the derived classes as per the need so that every derived class has the same function that performs specific (as redefined in the derived class) action.

Objectives:

After studying this unit you should be able to:

- discuss polymorphism
- explain types of polymorphism
- discuss function overloading
- describe virtual functions
- explain pure virtual functions
- compare function overloading and function overriding

8.2 Introduction to Polymorphism

Polymorphism means same content but different forms. The origin of the word polymorphism comes from two Greek words' poly (many) and morphos (form), together meaning multiform. In C++, by the concept of polymorphism the same program code can call different functions of different classes. For example let us consider a situation where you want to create a base class shape and want to derive classes such as rectangle, circle, and triangle from it. Let us suppose that all these classes have a member function named draw which draws the object on the screen. It will be easy if there is a common code so that you can draw all the shapes mentioned above using the same code and the shape to be drawn is decided at runtime. The code is shown below:

```
Shape *ptrarr[100]
for (int j=0;j<n;j++)
ptrarr[j]->draw();
```

This is a very desirable capability by which totally different functions are executed by the same function call. If the ptrarr is pointing to a rectangle, a rectangle is drawn. If it is pointing to a circle, a circle is drawn.

This is exactly what polymorphism is. However, we have to fulfill several conditions to implement this approach. The first condition is that all the classes i.e. rectangle, circle and triangle should be derived from a single class. The second condition is that you should declare draw function as virtual in the base class (in the shape class). Once you write application by using the polymorphism concept, then it can easily be extended, providing new objects that conform to the original interface. It is unnecessary to recompile original programs by adding new types. To exhibit the new changes along with the old application, only re-linking is required. This is the achievement of C++ object-oriented programming. In programming language, there has always been a need for adding and customizing. By using polymorphism in C++ time and effort is reduced and future maintenance is also made easy.

The advantages of polymorphism are:

- Helps in reusability of code.
- Provides easier maintenance of applications.
- Helps in achieving robustness in applications.

Program to demonstrate polymorphism

Program poly1.cpp

```
#include  <iostream.h>
// a base class declaration and the implementation part
  class   vehicle
  {
    int    wheels;
    float  weight;
   public:
  void  message(void)  // first message
   {
         cout<<"Vehicle message, from vehicle, the base class\n";}
       }
      };
        // a derived class declaration and implementation part
      class  car : public  vehicle
      {
       int   passenger_load;
      public:
         void   message(void)    // second message
          {
        cout<<"Car message, from car, the vehicle derived class\n";
      }

     };
      class truck: public vehicle
      {
      int passenger_load;
      float payload;
      public:
      int  passengers(void)
        {
         return  passenger_load;
        }
      };
      class boat: public vehicle
```

```
{
int  passenger_load;
public:
int  passengers(void)
  {
   return passenger_load;
  }
  void message (void)     // third message
   {
     cout<<"Boat message, from boat, the vehicle derived class\n";
    }
};
// the main program
int main()
{
vehicle unicycle;
car      sedan_car;
 truck    trailer;
 boat     sailboat;
   unicycle.message();
sedan_car.message();
trailer.message();
sailboat.message();
// base and derived object assignment
unicycle = sedan_car;
unicycle.message();
// system("pause");
return 0;
}
```

The output of the program is:

```
Vehicle message, from vehicle, the base class
Car message, from car, the vehicle derived class
Vehicle message, from vehicle, the base class
Boat message, from boat, the vehicle derived class
Vehicle message, from vehicle, the base class
```

Self Assessment Questions

1. _____ enables the same program code to call different functions of different classes.
2. Polymorphism in C++ reduces time, effort and by this makes future maintenance easy. (True/False)

8.3 Types of Polymorphism

There are two levels at which polymorphism can be achieved. These are:

1. Compile time polymorphism
2. Run time polymorphism

We will discuss these two briefly.

1. Compile time polymorphism

It is achieved through function overloading and operator overloading. You have already studied operator overloading in unit 6 and you will study function overloading in next section. In operator overloading the member functions of the same name but different arguments help the compiler to link the appropriate function definition to a specific function call during compile time. This type of binding is known as static or early binding. Early binding simply means that an object is bound to its function call at compile time. Hence, the polymorphism achieved through function overloading is called compile-time polymorphism.

2. Run-time polymorphism

This type of polymorphism is achieved through virtual functions. You will study virtual functions in the coming section of this unit. In this type of polymorphism binding or linking of a function definition to a specific function call is done during run-time. Hence, the polymorphism achieved through virtual functions is called run-time polymorphism.

At run time, when it is known objects of which class are under consideration, the appropriate version of the function is invoked, Since the function is linked with a particular class much later after the compilation, this process is termed as late binding. It is also known as dynamic binding because the selection of the appropriate function is dome dynamically at run time. Dynamic binding is one of the powerful features of C++. This requires the use of pointers to objects (already discussed in unit 4).

The following figure shows the levels of polymorphism.

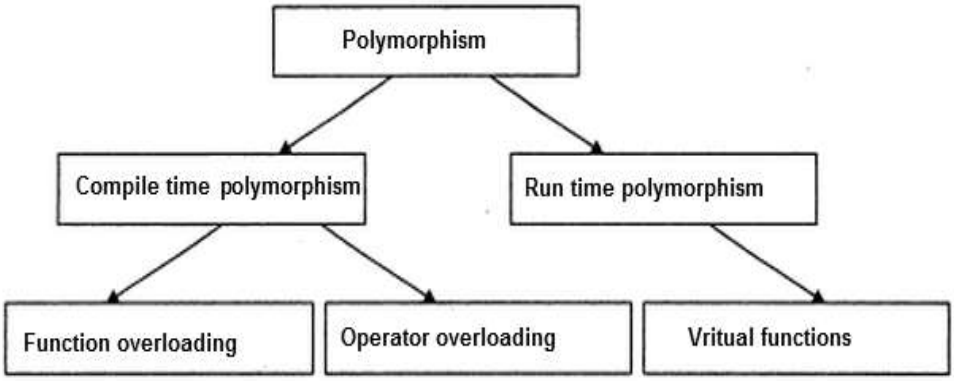

Figure 8.1: Achieving polymorphism

Self Assessment Questions

3. There are two types of polymorphism and these are _____ and _____ polymorphism.
4. _____ polymorphism is achieved through function overloading and operator overloading.
5. _____ polymorphism is achieved through virtual functions.
6. In operator overloading, the member functions of the same name but of different arguments help the compiler to link the appropriate function definition to a specific function call during compile time. This type of binding is known _____.
7. Dynamic binding requires the use of pointers to objects. (True/False)

8.4 Function Overloading

In C++ when a function has multiple declarations of the same function name in the same scope, it is called function overloading. These declarations differ in the type and number of arguments in the argument list. When an overloaded function is called, the types of actual arguments are compared with types of formal arguments. Thus, a correct function is selected. Overloaded functions enable programmers to supply different semantics for a function, depending on the types and number of arguments. The example of function overloading is shown below. In this program the function *area* is overloaded to calculate the area of circle, rectangle and triangle.

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#define pi 3.14
class Areacalculate
{
     public:
       void area(int);  //circle
       void area(int,int);  //rectangle
       void area(float,int,int);  //triangle
};
 void Areacalculate::area(int a)
{
     cout<<"Area of Circle:"<<pi*a*a;
}
void Areacalculate::area(int a,int b)
{
     cout<<"Area of rectangle:"<<a*b;
}
void Areacalculate::area(float t,int a,int b)
{
     cout<<"Area of triangle:"<<t*a*b;
}
 void main()
{
    int ch;
    int a,b,r;
    clrscr();
    fn obj;
    cout<<"\n\t\tFunction Overloading";
    cout<<"\n1.Area of Circle\n2.Area of Rectangle\n3.Area of
Triangle\n4.Exit\n:";
    cout<<"Enter your Choice:";
    cin>>ch;
```

```
    switch(ch)
   {
          case 1:
            cout<<"Enter Radious of the Circle:";
            cin>>r;
            obj.area(r);
            break;
          case 2:
            cout<<”Enter Sides of the Rectangle:”;
            cin>>a>>b;
            obj.area(a,b);
            break;
          case 3:
            cout<<"Enter Sides of the Triangle:";
            cin>>a>>b;
            obj.area(0.5,a,b);
            break;
          case 4:
            exit(0);
   }
getch();
}
```

The output of the above program is:

```
Function Overloading
1. Area of Circle
2. Area of Rectangle
3. Area of Triangle
4. Exit
Enter Your Choice: 2
Enter the Sides of the Rectangle: 5 5
Area of Rectangle is: 25
1. Area of Circle
2. Area of Rectangle
3. Area of Triangle
4. Exit
Enter Your Choice: 4
```

8.5 Introduction to Virtual Functions

Virtual means appearing like something while in reality it is something else.

Virtual function is a member function that you expect to be redefined in derived classes. 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. A virtual function is a function that must be overridden. That means it is declared in the base class using virtual keyword and its functionality is redefined by its derived classes.

As mentioned earlier, polymorphism refers to the property by which objects belonging to different classes are able to respond to the same message, but in different forms. An essential requirement of polymorphism is the ability to refer to objects without any regard to their classes. This necessitates the use of single pointer variable to refer to the objects of different classes. Here we use pointer to base class to refer all the derived objects. But it is found that the pointer of base class, even when it is made to contain the address of derived class, always executes the function in the base class. The contents of the pointer are ignored by the compiler and it chooses the member function that matches the type of the pointer. You may wonder how can one achieve polymorphism? It is achieved using virtual functions.

When the same function name is used in both base and derived classes, then the base class function is declared as virtual using the *virtual* keyword before the declaration of the function. When the function is declared as virtual then C++ can determine which function is to be used at run time based on the type of the object pointed to by the base pointer, rather than the type of the pointer. Thus, by making the base pointer to point to different objects, you can execute different versions of virtual functions.

Let us consider that there is a base class named as *base* and two classes derv1 and derv2 are derived *publicly* from class *base.* Suppose that you want to create a pointer that points to the objects of any of the derived class objects. If the pointer to derv1 is created then it will point to derv1 only. If you try to assign any other object to the pointer then it is not accepted by the compiler. The solution to this is to create pointer to the base class.

// objectptr.cpp

```
# include <iostream.h>
class base
{
 public:
 void show()
  {
    cout<<“base”<<endl;
   }
};
class derv1:public base
 {
  public:
 void show()
  {
    cout<<“derv1”<<endl;
   }
  };
class derv2: public base
 {
   public:
  void show()
   {
     cout<<“derv2”<<endl;
    }
   };
void main()
{
derv1 dv1;
derv2 dv2;
base *ptr;
ptr=&dv1;
ptr->show();
ptr=&dv2;
ptr->show();
}
```

Surprisingly, the output of the above program will be:
base
base

You can see in the program above that even though the pointer is assigned the address of the derived class, the base class function is executed by the compiler. Then as already been discussed, you have to make the base class function virtually to get the desired result. In the program shown below the show() function of the base class is made virtual by prefixing it with *virtual* keyword.

```
//virtual.cpp
# include <iostream.h>
class base
 {
  public:
  virtual void show()              // virtual function
   {
     cout<<“base”<<endl;
    }
  };
class derv1:public base
 {
  public:
void show()
{
  cout<<“derv1”<<endl;
 }
};
class derv2: public base
 {
  public:
  void show()
   {
    cout<<“derv2”<<endl;
```

```
  }
  };
void main()
{
derv1 dv1;
derv2 dv2;
base *ptr;
ptr=&dv1;
ptr->show();
ptr=&dv2;
ptr->show();
}
```

By declaring the base class function as virtual, we now get the output as:
derv1
derv2

As you can observe, the compiler decides which class function is to be called during runtime depending on the contents in the pointer. This is known as late binding or dynamic binding.

8.5.1 Pure virtual functions

Most of the times, the idea behind declaring a function (in the base class) virtual, is to stop its execution. The base class function is used very rarely to perform any task. Its role is only to serve as a placeholder. Then you may question why it is required to define (in detail) virtual functions? This leads to the idea of pure virtual functions or do-nothing functions. A pure virtual function is virtual function with no definition. You can declare a function as pure virtual by the following syntax:

virtual <return type> <function name> = 0;

here *virtual* is the keyword.

function name is the name of the function that is to be made purely virtual.

return type is the type of the data that virtual function returns.

For example: virtual void move() = 0;

Note that the function is initialized to zero. Here the assignment operator '=' has nothing to do with the assignment i.e. the value zero is not assigned to

anything. It is just to inform the compiler that the function will be pure and does not have any body.

You should note that no code is associated with this function. But what implementation it has done on the class in which it is declared? This class has a function, which cannot be executed. Hence, it is not possible for you to declare any object of this class. In other words the class becomes an abstract base class. The main objective of the abstract base class is to provide some traits to the derived classes and to create base pointer required for achieving run time polymorphism.

Let us see an example program:

```
#include <iostream,h>
class Exforsys
{
public:
virtual void example()=0; //Denotes pure virtual Function Definition
};
class Exf1:public Exforsys
{
public:
void example()
{
 cout << "Welcome";
}
};
 class Exf2:public Exforsys
{
public:
void example()
{
cout << "To Training";
}
};

void main()
{
        Exforsys* arra[2];
```

```
        Exf1 e1;
        Exf2 e2;
        arra[0]=&e1;
        arra[1]=&e2;
        arra[0]->example();
        arra[1]->example();
}
```

Output of the above program is:

WelcomeTo Training

In the above program example() is a pure virtual function and has no body and it is declared with notation =0. Exf1 and Exf2 are the two derived classes which are derived from the base class Exforsys. The pure virtual function example() takes up new definition. A list of pointers to the base class is defined in the main function. Here e1 and e2 are the objects of derived classes Exf1 and EXf2.

Two objects named e1 and e2 are defined for derived classes Exf1 and Exf2. The address of the objects e1 and e2 are stored in the array pointers which are then used for accessing the pure virtual function example() belonging to both the derived class EXf1 and EXf2.

Normally the virtual functions are declared in a base class and redefined in the derived class. The base class version of the function is not always used to perform the specific job or task, i.e. in case where the base class may be abstract with no possibility of declaration of an object, there is no sense in defining the virtual function in a base class. The actual function should be implemented or defined in each of the derived class. The base class function should just act as a placeholder to be overridden by the derived class versions. And it is not invoked by itself directly. To be more precise, a class having pure virtual function cannot be used to instantiate objects of its own. Hence, there is an error in the above program.

Self Assessment Questions

8. When a function has multiple declarations of the same function name in the same scope, it is called _____.

9. When the same function name is used in both base and derived classes, then the base class function is declared as virtual using the _____ keyword.
10. No code is associated with pure virtual functions. (True/ False).

8.6 Function Overloading v/s Function Overriding

You have already studied function overriding in the previous unit and function overloading in this unit. Now let us compare both the techniques.

- In both the techniques, same function name is used. But different parameters are passed in function overloading whereas the number of parameters passed to the function are same in function overriding.
- The function overloading may have different return type but in function overriding the return type of base and derived member function is the same.
- Function overloading is a method that allows defining multiple member functions with the same name but different signatures. The signature means the function name, number and types of parameters it accepts. Whereas, overriding is the process that allows the derived class to redefine the behavior of member functions which the derived class inherits from a base class. The signatures of both base class member function and derived class member function are the same; however the implementation and therefore the behavior will differ.
- Overloaded functions are in same scope whereas overridden functions are in different scope.

8.7 Summary

- Polymorphism means same content but different forms. By the concept of polymorphism the same program code can call different functions of different classes.
- There are two levels at which polymorphism can be achieved. These are:
  - Compile time polymorphism – It is achieved through function overloading and operator overloading and
  - Run time polymorphism – This type of polymorphism is achieved through virtual functions.

- In C++, when a function has multiple declarations of the same function name in the same scope, it is called function overloading. These declarations differ in the type and number of arguments in the argument list.
- Virtual function is a member function that you expect to be redefined in derived classes. 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 function is a member function that you expect to be redefined in derived classes. A pure virtual function is virtual function with no definition. It can be declared as follows: virtual <return type> <function name> = 0.

8.8 Terminal Questions

1. Imagine a publishing company markets books and CD-ROMS. Create a class called publication which stores titles and price of publications. Derive two classes: book which adds a page count and CD-ROM which adds playing time in minutes. Each of the three classes should have a getdata() and putdata() function to get its data from the user and display the data respectively. Write a main program that creates an array of pointers to publication and in a loop, ask the user for data for book or cdrom. When user has finished entering data, display all the data.
2. Describe the concept of polymorphism.
3. Describe the types of polymorphism.
4. Explain function overloading with an example.
5. Explain the concept of virtual function.
6. Discuss pure virtual functions.
7. Write the output of the following program:

```
#include <iostream.h>
class Shape
 {
  protected:
    int width, height;
  public:
    Shape(int a=0, int b=0)
```

```
    {
      width = a;
      height = b;
    }
   virtual int area()
    {
      cout << ”Parent class area :” <<endl;
      return 0;
    }
};
class Rectangle: public Shape{
  public:
    Rectangle( int a=0, int b=0):Shape(a, b) { }
    int area ()
    {
      cout << "Rectangle class area :" <<endl;
      return (width * height);
    }
};
class Triangle: public Shape{
  public:
    Triangle( int a=0, int b=0):Shape(a, b) { }
    int area ()
    {
      cout << "Triangle class area :" <<endl;
      return (width * height / 2);
    }
};
// Main function for the program
int main( )
{
  Shape *shape;
  Rectangle rec(10,7);
  Triangle  tri(10,5);

  // store the address of Rectangle
  shape = &rec;
```

```
    // call rectangle area.
    shape->area();

    // store the address of Triangle
    shape = &tri;
    // call triangle area.
    shape->area();

    return 0;
}
```

8. Write the ouptput of the following program:

```
#include<iostream.h>
class X
{
  public:
  void virtual display()=0;    //pure virtual function
};
class Y : public X  //class Y publically inherits class X
{
  public:
   void display()
   {
    cout<< " ABC" << endl;
  }
};
int main()
{
 X *xptr;
X objx;       //error
Y objy;
Xptr = &objy;
Xptr -> display();
return 0;
}
```

9. Compare function overloading and function overriding.

8.9 Answers

Self Assessment Questions

1. Polymorphism
2. True
3. Run-time, compile-time
4. Compile time
5. Run-time
6. Static or early binding
7. True
8. Function overloading
9. Virtual
10. True

Terminal Questions

1. //publish.cpp

```
# include <iostream.h>
#include<conio.h>
class publication
{ protected:
char title[80];
float price;
public:
virtual void getdata()
{ cout<<endl<<"enter title:";
cin>>title;
cout<<endl<<"enter price";
cin>>price;
}
virtual void putdata()
{
cout<< endl<<"Title"<<title;
cout<<endl<<"Price"<<price;
}
};
class book: public publication
{
private:
```

```
int pages;
public:
void getdata()
{ publication::getdata();
 cout<<endl<<"enter number of pages";
 cin>>pages;
}
void putdata()
{
publication::putdata();
cout<< endl<<"Number of pages"<<pages;
}
};
class cdrom: public publication
{
private:
float time;
public:
void getdata()
{ publication::getdata();
 cout<<endl<<"enter playing time:";
cin>>time;
}
void putdata()
{
publication::putdata();
cout<< endl<<"Playing time"<<time;
}
};
void main()
{
publication * ptr[10];
book* bptr;
cdrom* cptr;
char ch;
int n=0;
do
```

```
{
cout<<"Enter data for book or cdrom(b/c)";
cin>>ch;
if (ch=='b')
{
 bptr= new book;
bptr->getdata();
ptr[n++]=bptr;
}
else
{cptr=new cdrom;
cptr->getdata();
ptr[n++]=cptr;
}
cout<<" enter another (y/n)";
cin>>ch;
}while (ch=='y');
for(int j=0;j<n;j++)
ptr[j]->putdata();
getch();
} .
```

2. Polymorphism means the same content but different forms. The origin of the word polymorphism comes from two Greek words' poly (many) and morphos (form), together meaning multiform. In C++, by the concept of polymorphism the same program code can call different functions of different classes. Refer section 8.2 for more details.
3. There are two levels at which polymorphism can be achieved. These are:
   Compile time polymorphism - It is achieved through function overloading and operator overloading Run-time polymorphism – This type of polymorphism is achieved through virtual functions. Refer section 8.3 for more details.
4. In C++ when a function has multiple declarations of the same function name in the same scope, it is called function overloading. These declarations differ in the type and number of arguments in the argument list. Refer section 8.4 for more details.

5. Virtual means to appear like something while in reality it is something else. Virtual function is a member function that you expect to be redefined in derived classes. 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. Refer section 8.5 for more details.
6. Most of the times, the idea behind declaring a function (in the base class) virtual, is to stop its execution. The base class function is used very rarely to perform any task. Its role is only to serve as a placeholder. Then the question may arise as to why it is required to define (in detail) virtual functions? This leads to the idea of pure virtual functions or do-nothing functions. A pure virtual function is virtual function with no definition. Refer section 8.5 for more details.
7. Output is :
   Rectangle class area
   Triangle class area.
8. Output is :
   ABC.
9. In both the techniques same function name is used. But different parameters are passed in function overloading whereas the number of parameters passed to the function are same in function overriding.
   Refer section 8.6 for more details.

References:

- Oriented Programming With C++, by Subhash K U. Pearson Education India.
- Data Structures and Algorithms, first edition, by A. A. Puntambekar. Technical Publications.