Assignment No. 05

Roll No. SC55 - Shreyas Chavhan

===================================================

# Virtual Function (Polymorphism)

## Problem Statement -

Write C++ Program with base class convert declares two variables, val1

and val2, which hold the initial and converted values, respectively. It also

defines the functions getinit( ) and getconv( ), which return the initial value

and the converted value. Int Val1 = 5 Explicitly convert the int to float

These elements of convert are fixed and applicable to all derived classes that

will inherit convert. However, the function that will actually perform the

conversion, compute ( ), is a pure virtual function that must be defined by

the classes derived from convert. The specific nature of compute ( ) will be

determined by what type of conversion is taking place.

Baseclass getinit() getcnv()

## Theory -

Polymorphism in C++

The word polymorphism means having many forms. Typically, polymorphism occurs when there

is a hierarchy of classes and they are related by inheritance.

C++ polymorphism means that a call to a member function will cause a different function to be

executed depending on the type of object that invokes the function.

Virtual Function:

A virtual function is a function in a base class that is declared using the keyword virtual.

Defining in a base class a virtual function, with another version in a derived class, signals to the

compiler that we don&#39;t want static linkage for this function.

Late Binding

In Late Binding function call is resolved at runtime. Hence, now compiler determines the type of

object at runtime, and then binds the function call. Late Binding is also called Dynamic Binding

or Runtime Binding.

What we do want is the selection of the function to be called at any given point in the program to

be based on the kind of object for which it is called. This sort of operation is referred to as

dynamic linkage, or late binding.

Problem without Virtual Keyword

class Base

{

public:

void show()

{

cout &lt;&lt; &quot;Base class&quot;;

}

};

class Derived:public Base

{

public:

void show()

{

cout &lt;&lt; &quot;Derived Class&quot;;

}

}

int main()

{

Base\* b; //Base class pointer

Derived d; //Derived class object

b = &amp;d;

b-&gt;show(); //Early Binding Ocuurs

}

Output : Base class

When we use Base class&#39;s pointer to hold Derived class&#39;s object, base class pointer or reference

will always call the base version of the function

Using Virtual Keyword

We can make base class&#39;s methods virtual by using virtual keyword while declaring them. Virtual

keyword will lead to Late Binding of that method.

class Base

{

public:

virtual void show()

{

cout &lt;&lt; &quot;Base class&quot;;

}

};

class Derived:public Base

{

public:

void show()

{

cout &lt;&lt; &quot;Derived Class&quot;;

}

}

int main()

{

Base\* b; //Base class pointer

Derived d; //Derived class object

b = &amp;d;

b-&gt;show(); //Late Binding Ocuurs

}

Output : Derived class

On using Virtual keyword with Base class&#39;s function, Late Binding takes place and the derived

version of function will be called, because base class pointer pointes to Derived class object.

Pure Virtual Functions:

It&#39;s possible that you&#39;d want to include a virtual function in a base class so that it may be redefined

in a derived class to suit the objects of that class, but that there is no meaningful definition you

could give for the function in the base class.

class Shape

{

protected:

int width, height;

public:

Shape( int a=0, int b=0)

{

width = a;

height = b;

}

// pure virtual function

virtual int area() = 0;

};

The = 0 tells the compiler that the function has no body and above virtual function will be called

pure virtual function.

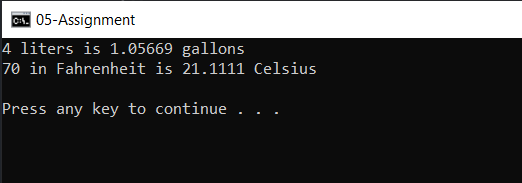
## Code -

|  |
| --- |
| /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Write C++ Program with base class convert declares two variables, val1  and val2, which hold the initial and converted values, respectively. It also  defines the functions getinit( ) and getconv( ), which return the initial value  and the converted value. Int Val1 = 5 Explicitly convert the int to float  These elements of convert are fixed and applicable to all derived classes that  will inherit convert. However, the function that will actually perform the  conversion, compute ( ), is a pure virtual function that must be defined by  the classes derived from convert. The specific nature of compute ( ) will be  determined by what type of conversion is taking place.  Baseclass getinit() getcnv()  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  #include <iostream>  using namespace std;  class convert  {  protected:  double val1;  double val2;  public:  convert(double i)  {  val1 = i;  }  double getconv()  {  return val2;  }  double getinit()  {  return val1;  }  virtual void compute() = 0;  };  class l\_to\_g : public convert  {  public:  l\_to\_g(double i) : convert(i)  {  }  void compute()  {  val2 = val1 / 3.7854;  }  };  // Fahrenheit to Celsius  class f\_to\_c : public convert  {  public:  f\_to\_c(double i) : convert(i)  {  }  void compute()  {  val2 = (val1 - 32) / 1.8;  }  };  int main()  {  convert \*p;  l\_to\_g lgob(4);  f\_to\_c fcob(70);  p = &lgob;  cout << p->getinit() << " liters is ";  p->compute();  cout << p->getconv() << " gallons\n";  p = &fcob;  cout << p->getinit() << " in Fahrenheit is ";  p->compute();  cout << p->getconv() << " Celsius\n";  return 0;  } |

Roll No. SC55 - Shreyas Chavhan

===================================================

## Output -



===================================================

Roll No. SC55 - Shreyas Chavhan