**Experiment No. 5**

**Title:** **Implementation of Dynamic Binding**

**Batch: B2 Roll No.: 16010421119 Experiment No.: 5**

**Aim**: Write a C++ program to create a class ‘shape’ which stores float values ‘dim1’ and ‘dim2’ as two dimensions of that shape. Function ‘read’ accepts these dimensions and function ‘area’ returns sum of dim1 and dim2 as the area of that shape. Derive class ‘triangle’ from shape which redefines area appropriately to return area of a triangle. Derive class ‘rectangle’ from shape which redefines area appropriately to return area of a rectangle. Write a program to make object pointers of class shape to point to objects of class triangle and rectangle respectively. Display area of each by calling the ‘area’ method with appropriate base pointer. Make proper provision so as to achieve dynamic binding.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Resources needed: Text Editor, C++ compiler**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

### Theory:

**Function Overriding**

If we inherit a class into the derived class and provide a definition for one of the base class's function again inside the derived class, then that function is said to be overridden, and this mechanism is called Function Overriding. For function overriding, function that is redefined must have exactly the same declaration in both base and derived class, that means same name, same return type and same parameter list.

**Virtual Functions**

Run time polymorphism is also called as Late Binding or Dynamic Binding. Run time polymorphism is achieved using virtual function. Virtual Function is a function in base class, which is overridden in the derived class, and which tells the compiler to perform Dynamic Binding on this function. ‘Virtual’ keyword is used to make a member function of the base class virtual. In Dynamic Binding function call is resolved at runtime. Hence, now compiler determines the type of object at runtime, and then binds the function call.

Without virtual function, when we use Base class's pointer to hold Derived class's object, base class pointer or reference will always call the base class version of the overridden function.

On using ‘virtual’ keyword with Base class's function, Dynamic Binding takes place and the derived version of function will be called, because base class pointer pointes to Derived class object.

**Pure Virtual Function**

A pure virtual function is a function that has no definition relative to base class.

Its syntax is as follws:

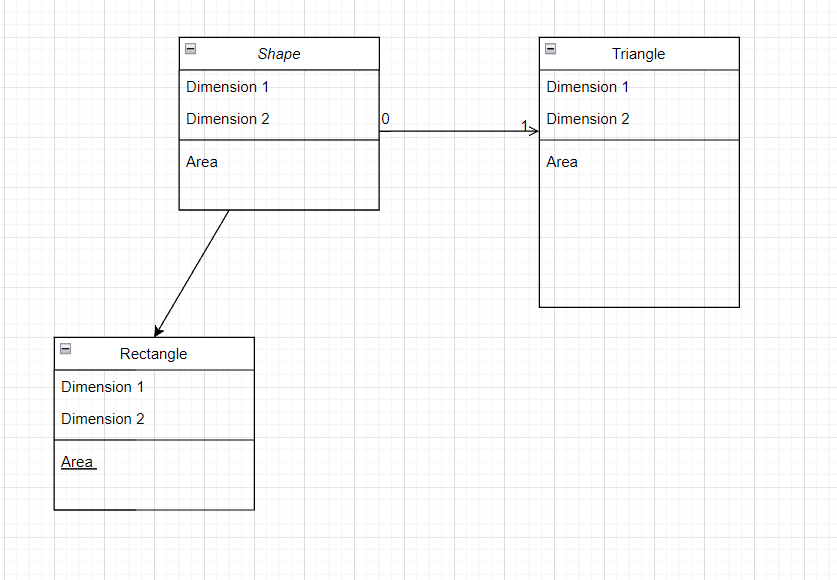
virtual return-type function-name(parameter list) = 0;

In such case each derived class should define the function or redeclare it as a pure virtual function.

**Abstract class**

A class having pure virtual function is called abstract class. An object of an abstract class cannot be created. A base class which is abstract is called abstract base class. Abstract base class is used to provide some traits to the derived classes and to create a base pointer required for run time polymorphism.

Class Diagram:



\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Results: (Program with snapshot of output)**

**Program:**

**#include <iostream>**

**using namespace std;**

**class Shape {**

**protected:**

**float dim1, dim2;**

**public:**

**void read(float i, float j=0)**

**{**

**dim1 = i;**

**dim2 = j;**

**}**

**virtual void area(void)**

**{**

**cout << "No area"<<endl;**

**}**

**} ;**

**class triangle : public Shape**

**{**

**public:**

**void area(void)**

**{**

**cout << "Triangle with dimensions "<<dim1<<" and "<<dim2<<endl;**

**cout << "Area = "<<dim1 \* 0.5 \* dim2<<"\n"<<endl;**

**}**

**};**

**class rectangle : public Shape**

**{**

**public:**

**void area(void)**

**{**

**cout << "Rectangle with dimensions "<<dim1<<" and "<<dim2<<endl;**

**cout << "Area = "<<dim1 \* dim2<<"\n"<<endl;**

**}**

**};**

**int main()**

**{**

**Shape \*p;**

**triangle t1;**

**triangle t2;**

**triangle t3;**

**rectangle r1;**

**rectangle r2;**

**p = &t1;**

**p->read(10.0, 5.0);**

**p->area();**

**p = &t2;**

**p->read(45.0, 12.0);**

**p->area();**

**p = &t3;**

**p->read(13.0, 11.0);**

**p->area();**

**p = &r1;**

**p->read(15.8, 13.6);**

**p->area();**

**p = &r2;**

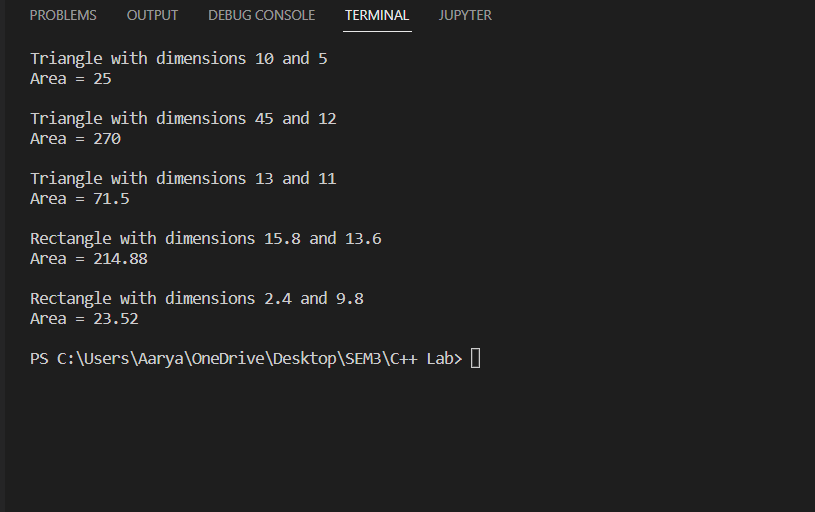
**p->read(2.4, 9.8);**

**p->area();**

**return 0;**

**}**

**Output:**

****

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Test Cases (minimum 5 test cases required):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr. No.** | **Sample Input** | **Sample Output** | **Description** | **Test Case Type (general/special)** | **Pass/Fail** |
| **1.** | **Dim1 = 10**  **Dim2 = 5** | **25**  **(Area)** | **Triangle** | **General** | **Pass** |
| **2.** | **Dim1 = 10**  **Dim2 = 5** | **270**  **(Area)** | **Triangle** | **General** | **Pass** |
| **3.** | **Dim1 = 10**  **Dim2 = 5** | **71.5**  **(Area)** | **Triangle** | **General** | **Pass** |
| **4.** | **Dim1 = 10**  **Dim2 = 5** | **214.88**  **(Area)** | **Rectangle** | **General** | **Pass** |
| **5.** | **Dim1 = 10**  **Dim2 = 5** | **23.52**  **(Area)** | **Rectangle** | **General** | **Pass** |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Questions:**

1. Compare early binding and late binding with a suitable example.

Ans:

Binding refers to the process of converting identifiers (such as variable and performance names) into addresses. Binding is done for each variable and functions. For functions, it means that matching the call with the right function definition by the compiler. It takes place either at compile time or at runtime.

Early Binding (compile-time time polymorphism) As the name indicates, compiler (or linker) directly associate an address to the function call. It replaces the call with a machine language instruction that tells the mainframe to leap to the address of the function.

Late Binding : (Run time polymorphism) In this, the compiler adds code that identifies the kind of object at runtime then matches the call with the right function definition (Refer this for details). This can be achieved by declaring a virtual function.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Outcomes:**

**CO 3.2 -**

Introduction to Virtual Functions, Abstract Base Classes And Concrete Classes, Polymorphism, New Classes And Dynamic Binding, Virtual Destructors, Pointers to virtual functions

**Conclusion: (Conclusion to be based on the outcomes achieved)**

We have learnt how to implement virtual functions with the example of shapes and other aspects where this can be used and put efficiently.

**Grade: AA / AB / BB / BC / CC / CD /DD**

Signature of faculty in-charge with date

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**References:**

**Books/ Journals/ Websites:**

1. E Balagurusamy, Object oriented Programming with C++, Tata McGraw-Hill, 8th Edition September 2020
2. Herbert Schildt, C++: The Complete Reference, McGraw Hill Education, 4th edition, July 2017
3. Jeff Langr, Modern C++ Programming with Test-Driven Development : Code Better,Sleep Better, O′Reilly, 1st edition, November 2013
4. <https://docs.microsoft.com/en-us/cpp/cpp/?view=msvc-170>