AST21105 Object-Oriented Programming & Design

Lab 12 – Virtual Function

A. Submission Details

In this lab, you are required to submit **ONE** C++ program with **SIX** files to solve the given problem shown in the section "Polygon Class". To make the program implementation easier, you are suggested to write your program using Visual Studio .NET instead of doing it directly using paper & pencil. After you have completed the implementation, submit your program files (i.e. Polygon,h, Polygon.cpp, Rectangle.h, Rectangle.cpp, Triangle.h and Triangle.cpp for this lab) using the electronic "drop-box" in Canvas, **One week** after your lab section is conducted. For details, please refer to the following:

Tuesday sections	*by*	*19:00 on 19 April 2016 (Tuesday)*
Wednesday sections	*by*	*19:00 on 20 April 2016 (Wednesday)*

You are reminded to double-check your solution to verify everything is correct before submission. You are also required to put your name and student ID at the beginning of your source file.

Important: You are only able to submit your work once.

B. Objective

The objective of this lab is to help you get familiar with virtual function that you have learned in class. The first part of this lab is meant to be a simple review of virtual functions. While the second part of this lab is a practical task that allows you to use all the techniques introduced.

C. Review

C++ Virtual Function

If there are member functions with same name in base class and derived class, virtual functions gives programmer capability to call member function of different class by a same function call depending upon different context. This feature in C++ programming is known as polymorphism which is one of the important feature of OOP.

If a base class and derived class has same function and if you write code to access that function using pointer of base class then, the function in the base class is executed even if, the object of derived class is referenced with that pointer variable. This can be demonstrated by an example.

```
#include <iostream>
using namespace std;
class B
{
      public:
          void display()
             { cout<<"Content of base class.\n"; }
};

class D : public B
{
      public:
          void display()
             { cout<<"Content of derived class.\n"; }
};

int main()
{
      B *b;
      D d;
      b->display();

      b = &d;       /* Address of object d in pointer variable */
      b->display();
      return 0;
}
```

Output:

```
Content of base class.
Content of base class.
```

Note: An object(either normal or pointer) of derived class is type compatible with pointer to base class. So, b = &d; is allowed in above program.

In above program, even if the object of derived class d is put in pointer to base class, display( ) of the base class is executed(member function of the class that matches the type of pointer).

Virtual Function

If you want to execute the member function of derived class then, you can declare display( ) in the base class virtual which makes that function existing in appearance only but, you can't call that function. In order to make a function virtual, you have to add keyword virtual in front of a function.

```
#include <iostream>
using namespace std;
class B
{
      public:
        virtual void display()          /* Virtual function */
              { cout<<"Content of base class.\n"; }
};

class D1 : public B
{
      public:
           void display()
              { cout<<"Content of first derived class.\n"; }
};

class D2 : public B
{
      public:
           void display()
              { cout<<"Content of second derived class.\n"; }
};

int main()
{
      B *b;
      D1 d1;
      D2 d2;
      b = &d1;
      b->display();     /* calls display() of class derived D1 */
      b = &d2;
      b->display();     /* calls display() of class derived D2 */
      return 0;
}
```

Output:

```
Content of first derived class.
Content of second derived class.
```

After the function of base class is made virtual, code b->display( ) will call the display( ) of the derived class depending upon the content of pointer.

In this program, display( ) function of two different classes are called with same code which is one of the example of polymorphism in C++ programming using virtual functions.

D. Polygon Class

In this part, you are required to:

i) Create a New Project and give your project the name Lab12.

ii) In this lab, you are required to create three classes **Polygon, Rectangle and Triangle** that has the following features.

The Polygon class is the base class and the Rectangle and Triangle are derived from Polygon.

The UML class diagrams given below:

Polygon
- noOfSide : int - isAllSideEqual : bool - sideLength: int*
+ Polygon() + Polygon(int n, bool s) + ~Polygon() + setsideLength(int* sl) : void + printsideLength() : void + totalsideLength() : int + area() virtual : int

where

- noOfSide: a data member to store the number of sides of the polygon.
- isAllSideEqual: a data member to store whether the length of sides are all the same.
- sideLength: a data member to store the length of each side.
- Polygon(): a default **constructor** that set the number of sides to 3, isAllSideEqual to false and create a dynamic integer array with the size equals to the number of sides and store the address of the array to the pointer sideLength.

- Polygon(int n, bool s): a **constructor** that set the number of sides to n, isAllSideEqual to s and create a dynamic integer array with the size equals to the number of sides and store the address of the array to the pointer sideLength. If the n received is less than 3, set it to 3. Also, set the default value of s to false.
- ~Polygon(): a **destructor**, which releases the memory allocated for sideLength.
- setsideLength(int* sl): a function that copy each value in sl to the dynamic array pointed by sideLength.
- printsideLength(): an output function that outputs the length of the sides of the polygon in one line.
- totalsideLength(): an function returns the total length of all the sides of the polygon.
- area(): a virtual function that return 0 in the Polygon class.

Triangle
- width : int - height : int
+ Triangle(int w, int h, bool s) + area() virtual : int

where

- width: a data member to store the width of the triangle.
- height: a data member to store the height of the triangle.
- Triangle(int w, int h, bool s): a **constructor** that set the width and height. Also, pass 3 and s to the base class, Polygon, to initialize the noOfSide and isAllSideEqual.
- area(): a virtual function that calculate and return the area of the triangle by the width and height.

Rectangle
- width : int - height : int
+ Rectangle(int w, int h) + printsideLength(): void + area() virtual : int

where

- width: a data member to store the width of the rectangle.
- height: a data member to store the height of the rectangle.

- Rechtangle(int w, int h): a **constructor** that set the width and height. Also, check whether w and h are equal. If they are equal, set the isAllSideEqual to true. Otherwise, set the isAllSideEqual to false. Then, create a dynamic integer array with the size equals to 4 and store the length (width and height) of each side. Make the sideLength points to the dynamic array.
  Hint: You can create an array with the size of 4 and stored the width and height in it. Then call the setsideLength to initialize the array pointed by the sideLength pointer.
- printsideLength(): an output function that outputs the width and height of the rectangle in one line.
- area(): a virtual function that calculate and return the area of the rectangle by the width and height.

iii) Amain function (in main.cpp) has been provided for you to test out your class.

The following is the sample output.

```
Rectangle 1:
        Area: 20
        Side: 4 5 4 5
        Width = 4, Height = 5
        Total Side Length: 18
Rectangle 2:
        Area: 9
        Side: 3 3 3 3
        Width = 3, Height = 3
        Total Side Length: 12
Triangle:
        Area: 8
        Side: 5 4 3
        Total Side Length: 12
```

Marking Scheme:

Graded items	Weighting
1. Correctness of program (i.e. whether your code is implemented in a way according to the requirements as specified.)	60%
2. Indentation	30%
3. Documentation (with reasonable amount of comments embedded in the code to enhance the readability.)	10%
	100%

Program Submission Checklist

Before submitting your work, please check the following items to see you have done a decent job.

Items to be checked	☑ / ☒
1. Did I put my name and student ID at the beginning of all the source files?	☐
2. Did I put reasonable amount of comments to describe my program?	☐
3. Are they all in .cpp extension and named according to the specification?	☐
4. Have I checked that all the submitted code are compliable and run without any errors?	☐
5. Did I zip my source files using Winzip / zip provided by Microsoft Windows? Also, did I check the zip file and see if it could be opened? ***(Only applicable if the work has to be submitted in zip format.)***	☐
6. Did I submit my lab assignment to Canvas?	☐

-End-