

Laboratory Exercise 14 (Inheritance and Polymorphism)

Topics: Hierarchical Class Structures and Polymorphic Containers

Goals: Upon successful completion of this lab you should be able to:

1. Define a hierarchical class structure
2. Define Virtual Functions
3. Write application programs that use polymorphism
4. Create a doubly linked list of pointers to polymorphic objects.
5. Write a program that uses a doubly linked list of pointers to polymorphic objects

Related text sections:

Chapter 14

Chapter 15

Chapter 17

Laboratory Exercise 14 Instructions (part 1)

1. Create a new folder for Lab 14 and copy the files in `~tiawatts/cs215pickup/Lab14` to your new folder. The files are listed at the end of this lab.

2. Use the makefile to compile the program. Execute the program using the following input:

```
option: a
side: 3
option: b
one leg: 3
the other leg: 4
option: q
```

Your interactive session should look like:

```
Select one of the following options:
  a. Create an Equilateral Triangle
  b. Create a Right Triangle
  q. Exit
a
Enter the length of the side of the equilateral triangle: 3
The area of your 3 sided polygon is 0; its perimeter is 0.
Select one of the following options:
  a. Create an Equilateral Triangle
  b. Create a Right Triangle
  q. Exit
b
Enter the length of one leg of the right triangle: 3
Enter the length of the other leg of the right triangle: 4
The area of your 3 sided polygon is 0; its perimeter is 0.
Select one of the following options:
  a. Create an Equilateral Triangle
  b. Create a Right Triangle
  q. Exit
q
Good bye
```

3. The zero values in the output indicate that the functions to calculate the area and perimeters of the equilateral and right triangles have not been completed. Using the formulas you found for the homework assignment, complete the Area and Perimeter functions for the Equilateral and Right classes.
4. Recompile and execute the program. The correct output should be:
The area of your 3 sided polygon is 3.89711; its perimeter is 9.
The area of your 3 sided polygon is 6; its perimeter is 12.
5. Using the equilateral and right triangle classes as a model, complete the implementation of the scalene triangle class. Include a menu option (c) for a scalene triangle. Include a case in the create function for a scalene triangle.
6. Compile and test your modifications.

7. Add two new classes to the polygon heirarchy: Square and Rectangle. These new classes should both be children of the Quadrilateral class. Add options (d and e) to the menu and cases to the create function to incorporate these new classes in the application program.
8. Complete the implementation of the Regular polygon class. Include the Regular polygon class in the implementation program (option f).
9. When you are convinced that your polygon class heirarchy is working correctly, copy your well documented .h and .cpp files to the dropbox as *yourlastnameL14.h* and *yourlastnameL14.cpp* respectively.

makefile

```
lab14 : Lab14.o polygon.o
        g++ -o lab14 Lab14.o polygon.o -g

Lab14.o : Lab14.cpp polygon.h
        g++ -c Lab14.cpp -g

polygon.o : polygon.cpp polygon.h
        g++ -c polygon.cpp -g

clean :
        rm -f core.* *.o lab14
```

Lab14.cpp

```
#include <iostream>
#include "polygon.h"

char menu ();

int main ()
{
    char option;
    do
    {
        option = menu ();
        Polygon * pp = create (option);
        if (pp)
        {
            std::cout << *pp;
            delete pp;
        }
    } while (option != 'Q');
    return 0;
}

char menu ()
{
    char choice;
    std::cout << "Select one of the following options:\n";
    std::cout << "\ta. Create an Equilateral Triangle\n";
    std::cout << "\tb. Create a Right Triangle\n";
    std::cout << "\tq. Exit\n";
    std::cin >> choice;
    return toupper(choice);
}
```

```

Polygon * create (char choice)
{
    Polygon * pp = NULL;
    switch (choice)
    {
        case 'A':
        {
            int side;
            std::cout << "Enter the length of the side of the "
                << "equilateral triangle: ";
            std::cin >> side;
            pp = new Equilateral (side);
            break;
        }
        case 'B':
        {
            int side1, side2;
            std::cout << "Enter the length of one leg of the "
                << "right triangle: ";
            std::cin >> side1;
            std::cout << "Enter the length of the other leg of the "
                << "right triangle: ";
            std::cin >> side2;
            pp = new Right (side1, side2);
            break;
        }
        case 'Q':
        {
            std::cout << "Good bye\n";
            break;
        }
        default:
            std::cout << "Invalid option - please try again\n";
    }
    return pp;
}

```

polygon.h

```

#ifndef POLYGON_H
#define POLYGON_H

#include <iostream>
using namespace std;

class Polygon
{
public:
    Polygon ();
    Polygon (const Polygon & p);
    ~Polygon ();
    Polygon & operator = (const Polygon & p);
    virtual int Sides () const { return 0; }
    virtual double Area () const { return 0; }
    virtual double Perimeter () const { return 0; }
    friend ostream & operator << (ostream & outs, const Polygon & P);
protected:
    double side;
private:
};

```

```

class Triangle : public Polygon
{
    public:
        Triangle ();
        Triangle (const Triangle & p);
        ~Triangle ();
        Triangle & operator = (const Triangle & p);
        int Sides () const;
    private:
};

class Equilateral : public Triangle
{
    public:
        Equilateral ();
        Equilateral (double S);
        ~Equilateral ();
        Equilateral (const Equilateral & p);
        Equilateral & operator = (const Equilateral & p);
        double Area () const;
        double Perimeter () const;
    private:
};

class Right : public Triangle
{
    public:
        Right ();
        Right (double S1, double S2);
        ~Right ();
        Right (const Right & p);
        Right & operator = (const Right & p);
        double Area () const;
        double Perimeter () const;
    private:
        double side2;
};

class Scalene : public Triangle
{
    public:
        Scalene ();
        Scalene (double S1, double S2, double S3);
        ~Scalene ();
        Scalene (const Scalene & p);
        Scalene & operator = (const Scalene & p);
        double Area () const;
        double Perimeter () const;
    private:
        double side2, side3;
};

class Quadrilateral : public Polygon
{
    public:
        Quadrilateral ();
        Quadrilateral (const Quadrilateral & p);
        ~Quadrilateral ();
        Quadrilateral & operator = (const Quadrilateral & p);
        int Sides () const;
    private:
};

```

```

// Add Square and Rectangle classes here

class Regular : public Polygon
{
    public:
        Regular ();
        Regular (const Regular & p);
        ~Regular ();
        Regular & operator = (const Regular & p);
        int Sides () const;
        double Area () const;
        double Perimeter () const;
    private:
        int num_sides;
};

#endif

```

polygon.cpp

```

include <iostream>
#include <cmath>
#include "polygon.h"
using namespace std;

Polygon::Polygon ()
{
    side = 0;
}

Polygon::Polygon (const Polygon & p)
{
    side = p.side;
}

Polygon::~Polygon ()
{
}

Polygon & Polygon::operator = (const Polygon & p)
{
    side = p.side;
    return * this;
}

ostream & operator << (ostream & outs, const Polygon & P)
{
    outs << "The area of your " << P.Sides()
        << " sided polygon is " << P.Area ()
        << "; its perimeter is " << P.Perimeter() << ".\n";
    return outs;
}

Triangle::Triangle ()
{
    side = 0;
}

Triangle::Triangle (const Triangle & p)
{
    side = p.side;
}

```

```

Triangle::~Triangle ()
{
}

Triangle & Triangle::operator = (const Triangle & p)
{
    side = p.side;
    return * this;
}

int Triangle::Sides () const
{
    return 3;
}

Equilateral::Equilateral ()
{
    side = 0;
}

Equilateral::Equilateral (double S)
{
    side = S;
}

Equilateral::~Equilateral ()
{
}

Equilateral::Equilateral (const Equilateral & p)
{
    side = p.side;
}

Equilateral & Equilateral::operator = (const Equilateral & p)
{
    side = p.side;
    return * this;
}

double Equilateral::Area () const
{
    // Calculate and return area here
    return 0;
}

double Equilateral::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

Right::Right ()
{
    side = side2 = 0;
}

Right::Right (double S1, double S2)
{
    side = S1;
    side2 = S2;
}

```

```

Right::~~Right ()
{
}

Right::Right (const Right & p)
{
    side = p.side;
    side2 = p.side2;
}

Right & Right::operator = (const Right & p)
{
    side = p.side;
    side2 = p.side2;
    return * this;
}

double Right::Area () const
{
    // Calculate and return area here
    return 0;
}

double Right::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

Scalene::Scalene ()
{
    side = side2 = side3 = 0;
}

Scalene::Scalene (double S1, double S2, double S3)
{
    side = S1;
    side2 = S2;
    side3 = S3;
}

Scalene::~~Scalene ()
{
}

Scalene::Scalene (const Scalene & p)
{
    side = p.side;
    side2 = p.side2;
    side3 = p.side2;
}

Scalene & Scalene::operator = (const Scalene & p)
{
    side = p.side;
    side2 = p.side2;
    side3 = p.side2;
    return * this;
}

double Scalene::Area () const
{

```

```

        // Calculate and return area here
        return 0;
    }

double Scalene::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

Quadrilateral::Quadrilateral ()
{
}

Quadrilateral::Quadrilateral (const Quadrilateral & p)
{
}

Quadrilateral & Quadrilateral::operator = (const Quadrilateral & p)
{
    return * this;
}

int Quadrilateral::Sides () const
{
    return 4;
}

Regular::Regular ()
{
}

Regular::Regular (const Regular & p)
{
}

Regular & Regular::operator = (const Regular & p)
{
    return * this;
}

int Regular::Sides () const
{
    return num_sides;
}

double Regular::Area () const
{
    // Calculate and return area here
    return 0;
}

double Regular::Perimeter () const
{
    // Calculate and return perimeter here
    return 0;
}

```

Laboratory Exercise 14 Instructions (part 2)

1. Copy the file LList2.tmp from your Lab 12 folder to your new folder. Modify the application program (Lab14.cpp) to include your LList2.tmp file. Modify the makefile to include LList2.tmp as a dependency for creating Lab14.o.
2. Instantiate a linked list of polygon pointers and modify the do while loop in the main function of the application:

```
    LList2 <Polygon *> PLP;
    do
    {
        option = menu ();
        Polygon * pp = create (option);
        if (pp)
            PLP.InsertLast (pp);
    } while (option != 'Q');
```

3. Add statements after the loop to print the contents of the linked list from beginning to end.
4. Use the makefile to compile the program. Execute the program using the following input:
option: a
side: 3
option: b
one leg: 3
the other leg: 4
option: q

The output should be:

```
The area of your 3 sided polygon is 3.89711; its perimeter is 9.
The area of your 3 sided polygon is 6; its perimeter is 12.
```

If your output looks more like:

```
0x804c290 0x804c2b0
```

than the correct output, you are printing the addresses of the objects instead of the data in the objects. Fix your program to print the data.

Test the other classes in your polygon class heirarchy.

5. Add a segment of code to print the data in reverse order.
6. Add a segment of code to find and print the total of the areas of the polygons in the list.
7. Add a segment of code to find and print the total of the perimeters of the polygons in the list.
8. When you are convinced that your application program is working correctly, copy your well documented .cpp file to the dropbox as *yourlastnameL14.cpp*.