C++ Lab 08 - Class Inheritance in C++

2.680 Unmanned Marine Vehicle Autonomy, Sensing and Communications

Ten Short CPP Labs

IAP 2024

Michael Benjamin, mikerb@mit.edu
Department of Mechanical Engineering
MIT, Cambridge MA 02139

1 Lab Eight Overview and Objectives

In this lab C++ class inheritance (sub-classing) is explored. This is one of the key capabilities in object-oriented program and is a prime distinction between C and C++. In MOOS-IvP it is also a key concept since all MOOS apps inherit from a MOOSApp superclass and all IvP Helm behaviors inherit from a IvPBehavior superclass. In this lab we will build a couple simple subclasses of the `SegList` class used previously in Lab 05.

In this lab the following topics are covered.

- C++ Subclasses
- Polymorphism
- Encapsulation
- Function Overloading
- Virtual Functions

2 Subclasses and Inheritance in C++

Class inheritance is a key part of C++ and object oriented languages in general. It allows for a natural way of organizing code and a form of *code re-use* in functions higher in the hierarchy usable by sub-classes. It complements the *has-a* relationship with the *is-a* relationship. The following couple links provide a good introduction:

- You can skip Section 11.7 on Multiple Inheritance
  http://www.learncpp.com/cpp-tutorial/111-introduction-to-inheritance
- You can skim or skip the sections on Friend Classes and Multiple Inheritance.
  http://www.cplusplus.com/doc/tutorial/inheritance

After you've read the above link(s), you should be ready for a couple short exercises.

2.1 Exercise 1: Creating a Simple Subclass of the SegList Class

Create two subclasses of the `SegList` class created in lab 05, called `Triangle` and `Square`. For now, these two classes will do nothing more than overload the `getSpec()` function. The overloaded function should invoke the `getSpec()` function already implemented in the `SegList` superclass. The overloaded function should simply add the type to the output of the string spec.

HINT: To see how to invoke a superclass function from within a subclass function, it's worth taking an extra careful read of the section *Adding to Existing Functionality* in Section 11.6 in the `learncpp.com` URL above.

In this exercise we will test the two new classes with a simple `main()` function that creates a `Triangle` instance and `Square` instance with vertices:

- `(0,0)`, `(10,0)`, and `(5,5)`
- `(0,0)`, `(10,0)`, `(10,10)`, and `(0,10)`

After instantiating and populating two instances, with the above vertices, just output the result of `getSpec()` to the terminal.

Call your files `Triangle.cpp/h`, `Square.cpp/h` and `shapes_v1.cpp` and build it to the executable `shapes_v1`. When your program runs it should be invocable from the command line with:

```
$ ./shapes_v1
Triangle: type=triangle,x=0,y=0,x=10,y=0,x=5,y=5
Square:   type=square,x=0,y=0,x=10,y=0,x=10,y=10,x=0,y=10
```

The solution to this exercise is in Section 4.1.

Building shapes_v1: From the previous lab on Build Systems and Makefiles, you should be able to make a simple Makefile for this exercise. If you'd like to peek ahead at a Makefile for all five exercises in this lab, see Section 4.6.

2.2 Exercise 2: Using Inherited Superclass Functions

In this exercise, add a function to the `SegList` superclass to calculate the "perimeter" of the line segements, including the distance from the end vertex back to the first vertex. This short exercise demonstrates a form of code re-use through inheritence. Call your new function `getPerimeter()` and implement it on a copy of the `SegList` class called `SegListV2`. You will also need to slightly modify your `Triangle` and `Square` classes to use the new superclass, so call them `TriangleV2` and `SquareV2` respectively. Your `main()` function in `shapes_v2.cpp` will only be modified to show the newly calculated perimeter on the same example shapes as the first exercise. When your program runs it should be invocable from the command line with:

```
$ ./shapes_v2
Triangle:   type=triangle,x=0,y=0,x=10,y=0,x=5,y=5
 Perimeter: 24.1421
Square:     type=square,x=0,y=0,x=10,y=0,x=10,y=10,x=0,y=10
 Perimeter: 40
```

The solution to this exercise is in Section 4.2.

Building shapes_v2: From the previous lab on Build Systems and Makefiles, you should be able to make a simple Makefile for this exercise. If you'd like to peek ahead at a Makefile for all five exercises in this lab, see Section 4.6.

3 Polymorphism

The term *polymorphism* refers to the ability to have several functions, with the same name, do different things depending on which class is invoking the function. A simple example of polymorphism:

```
1 + 2       // Integer addition
1.1 + 2.2   // Floating point addition
"1" + "2"   // String concatenation
```

Another example is when different classes have the same function defined, even with same semantic meaning, but behavior differently depending on the class. For example, in our next exercise we define the `getArea()` function on both the `Triangle` and `Square` classes, but the function is implemented differently to reflect the different way that area is calculated. Before proceeding to this exercise, have a read of:

- http://www.cplusplus.com/doc/tutorial/polymorphism

3.1 Exercise 3: Using Polymorphism in our Shapes Example

In this exercise, implement the `getArea()` function on both the `Shape` and `Triangle` class. In both functions, a handy utility would be the ability to know the min and max x-y values of the vertices. To this end, implement the following four utility functions at the superclass level:

- `double getMinX() const;`
- `double getMaxX() const;`
- `double getMinY() const;`
- `double getMaxY() const;`

Note that they are `const` functions since they don't alter the state of the object, only report on existing properties. After you have implemented these utility functions, implementing the `getArea()` function in both the `Triangle` and `Square` classes should be much simpler.

Implement your new `getArea()` functions on copies of the existing classes, now called `TriangleV3.cpp/h` and `SquareV3.cpp/h`. Implement your `getMin*()` utility functions on a copy of the existing superclass, now called `SegListV3.cpp/h`. Your new `main()` function should be in `shapes_v3.cpp`, and will only be modified to show the newly calculated area on the same example shapes as the first exercises. When your program runs it should be invocable from the command line with:

```
$ ./shapes_v3
Triangle: type=triangle,x=0,y=0,x=10,y=0,x=5,y=5
    Area: 25
Square:   type=square,x=0,y=0,x=10,y=0,x=10,y=10,x=0,y=10
    Area: 100
```

The solution to this exercise is in Section 4.3.

Building shapes_v3: From the previous lab on Build Systems and Makefiles, you should be able to make a simple Makefile for this exercise. If you'd like to peek ahead at a Makefile for all five exercises in this lab, see Section 4.6.

3.2 Exercise 4: Implementing Virtual Functions in our Shapes Example

In this exercise we'll make a small but key addition to the problem presented in the previous exercise. The goal is to be able to deal with a mixture of both `Triangle` and `Square` instances in a

generic manner since they both have the same superclass. For example, we should be able to have a container, such as a `vector` or `list`, of such instances and add up all their areas, not caring whether any of the instances is a `Triangle` or `Square`. In these cases the containers will contain *pointers to the baseclass*. For example:

- `vector<SegList*> my_shapes;`

In our example so far, it *is* possible to create the above type of container, but if one were to try to invoke the `getArea()` function on any of the instances, it would be disallowed since this function is not defined on the `SegList` base class. To see this, consider the below code snippet. This snippet assumes a valid `Triangle` and `Square` instance have been created as with all our prior exercises.

```
SegListV4 *shape1 = &triangle;
SegListV4 *shape2 = &square;

cout << "Triangle: " << shape1->getSpec() << endl;
cout << "    Area: " << shape1->getArea() << endl;
cout << "Square:   " << shape2->getSpec() << endl;
cout << "    Area: " << shape2->getArea() << endl;
```

In this snippet two base-class pointers are created, `shape1` and `shape2`, pointing to the `Triangle` and `Square` instance respectively. This is fine, but if you tried to do this the compiler would complain about the following two lines:

```
cout << "    Area: " << shape1->getArea() << endl;
cout << "    Area: " << shape2->getArea() << endl;
```

This is because the `getArea()` function is not defined (yet) on the base class. To overcome this, we can define the `getArea()` function as a *virtual function* at the base class level. A discussion on virtual functions can be found a the following URLs:

- http://www.cplusplus.com/doc/tutorial/polymorphism

Your job in this exercise is to add the `getArea()` function as a virtual function in the superclass. This modified superclass will be called `SegListV4.cpp/h`. Also create `TriangleV4.cpp/h` and `SquareV4.cpp/h` to use this new superclass. Confirm that the above snippet works in your new `shapes_v4.cpp` function. When your program runs it should be invocable from the command line with:

```
$ ./shapes_v4
Triangle: x=0,y=0,x=10,y=0,x=5,y=5
    Area: 25
Square:   x=0,y=0,x=10,y=0,x=10,y=10,x=0,y=10
    Area: 100
```

The solution to this exercise is in Section 4.4.

Building shapes_v4: From the previous lab on Build Systems and Makefiles, you should be able to make a simple Makefile for this exercise. If you'd like to peek ahead at a Makefile for all five exercises in this lab, see Section 4.6.

3.3 Exercise 5: One Final Improvement to our Shapes Example

You may have noticed that the output from the `getSpec()` function in the previous example is slightly different:

```
Triangle: x=0,y=0,x=10,y=0,x=5,y=5
Square:   x=0,y=0,x=10,y=0,x=10,y=10,x=0,y=10
```

Whereas before, in Exercise 3, it looked like:

```
Triangle: type=triangle,x=0,y=0,x=10,y=0,x=5,y=5
Square:   type=square,x=0,y=0,x=10,y=0,x=10,y=10,x=0,y=10
```

Do you see the difference? Before addressing why this change occurred, and fixing it here in this exercise, see if you can determine what caused the change before reading on....

Once we start regarding our shapes generically, as in the previous exercise, with pointers to the base class, we need to be aware that a invocation to a function like `SegList::getSpec()` will invoke exactly that function, not the `Subclass::getSpec()` function. Unless the base class function is virtual. Try making the `getSpec()` function in the base class virtual, and notice that the output reverts to again include the `type` information.

The solution to this exercise is in Section 4.5. It simply shows the one line change to `SegListV4.h` to make the `getSpec()` function in the base class to be now virtual.

4 Solutions to Exercises

4.1 Solution to Exercise 1

```
*-------------------------------------------------------------------------*/
/*  FILE:  shapes_v1.cpp (Eighth C++ Lab Exercise 1)                       */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/shapes_v1.cpp               */
/*  RUN:   shapes_v1                                                       */
/*-------------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include "Triangle.h"
#include "Square.h"

using namespace std;

int main()
{
  Triangle triangle;
  Vertex t_vertex1(0,0);
  Vertex t_vertex2(10,0);
  Vertex t_vertex3(5,5);
  triangle.addVertex(t_vertex1);
  triangle.addVertex(t_vertex2);
  triangle.addVertex(t_vertex3);

  Square square;
  Vertex s_vertex1(0,0);
  Vertex s_vertex2(10,0);
  Vertex s_vertex3(10,10);
  Vertex s_vertex4(0,10);
  square.addVertex(s_vertex1);
  square.addVertex(s_vertex2);
  square.addVertex(s_vertex3);
  square.addVertex(s_vertex4);

  cout << "Triangle: " << triangle.getSpec() << endl;
  cout << "Square:   " << square.getSpec() << endl;

  return(0);
}
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  Triangle.h                                                      */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Triangle.h                  */
/*-------------------------------------------------------------------------*/

#ifndef TRIANGLE_HEADER
#define TRIANGLE_HEADER

#include "SegList.h"

class Triangle : public SegList {
 public:
  Triangle() {};
  ~Triangle() {};

  std::string getSpec() const;   // recall const just means that this function
};                               // won't alter the state of an instance.

#endif
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  Triangle.cpp                                                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Triangle.cpp                */
/*-------------------------------------------------------------------------*/

#include "Triangle.h"

using namespace std;

//---------------------------------------------------------------------------
// Procedure: getSpec()

string Triangle::getSpec() const
{
  string spec = "type=triangle,";
  spec += SegList::getSpec();
  return(spec);
}
```

```
/*-----------------------------------------------------------------------*/
/*  FILE:  Square.h                                                      */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Square.h                  */
/*-----------------------------------------------------------------------*/

#ifndef SQUARE_HEADER
#define SQUARE_HEADER

#include "SegList.h"

class Square : public SegList {
 public:
  Square() {};
  ~Square() {};

  std::string getSpec() const;
};

#endif
```

```
/*-----------------------------------------------------------------------*/
/*  FILE:  Square.cpp                                                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Square.cpp                */
/*-----------------------------------------------------------------------*/

#include "Square.h"

using namespace std;

//---------------------------------------------------------------------
// Procedure: getSpec()

string Square::getSpec() const
{
  string spec = "type=square,";
  spec += SegList::getSpec();
  return(spec);
}
```

4.2 Solution to Exercise 2

```
/*-------------------------------------------------------------------------*/
/*  FILE:  shapes_v2.cpp (Eighth C++ Lab Exercise 2)                       */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/shapes_v2.cpp               */
/*  RUN:   shapes_v2                                                       */
/*-------------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include "TriangleV2.h"
#include "SquareV2.h"

using namespace std;

int main()
{
  TriangleV2 triangle;
  Vertex t_vertex1(0,0);
  Vertex t_vertex2(10,0);
  Vertex t_vertex3(5,5);
  triangle.addVertex(t_vertex1);
  triangle.addVertex(t_vertex2);
  triangle.addVertex(t_vertex3);

  SquareV2 square;
  Vertex s_vertex1(0,0);
  Vertex s_vertex2(10,0);
  Vertex s_vertex3(10,10);
  Vertex s_vertex4(0,10);
  square.addVertex(s_vertex1);
  square.addVertex(s_vertex2);
  square.addVertex(s_vertex3);
  square.addVertex(s_vertex4);

  cout << "Triangle:   " << triangle.getSpec() << endl;
  cout << " Perimeter: " << triangle.getPerimeter() << endl;    // Added line
  cout << "Square:     " << square.getSpec() << endl;
  cout << " Perimeter: " << square.getPerimeter() << endl;      // Added line

  return(0);
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SegListV2.h                                                     */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV2.h                 */
/*-------------------------------------------------------------------------*/

#ifndef SEGLIST_V2_HEADER
#define SEGLIST_V2_HEADER

#include <string>
#include <vector>
#include "Vertex.h"

class SegListV2 {
 public:
  SegListV2() {};
  ~SegListV2() {};

  void addVertex(Vertex vertex) {m_vertices.push_back(vertex);};

  std::string getSpec() const;

  double getPerimeter() const;

 protected:
  std::vector<Vertex> m_vertices;
};
#endif
```

```
/*--------------------------------------------------------------------*/
/*  FILE:  SegListV2.cpp                                              */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV2.cpp           */
/*--------------------------------------------------------------------*/

#include <cmath>           // For including the hypot function
#include "SegListV2.h"

using namespace std;

//--------------------------------------------------------------------
// Procedure: getSpec

string SegListV2::getSpec() const
{
  string spec;
  for(unsigned int i=0; i<m_vertices.size(); i++) {
    if(i != 0)
      spec += ",";
    spec += m_vertices[i].getSpec();
  }

  return(spec);
}


//--------------------------------------------------------------------
// Procedure: getPerimeter

double SegListV2::getPerimeter() const
{
  double total = 0;
  for(unsigned int i=0; i<m_vertices.size(); i++) {
    unsigned int next_ix = i+1;
    if((i+1) == m_vertices.size())
      next_ix = 0;

    double xdelta = m_vertices[i].getX() - m_vertices[next_ix].getX();
    double ydelta = m_vertices[i].getY() - m_vertices[next_ix].getY();
    double dist = hypot(xdelta, ydelta);
    total += dist;
  }

  return(total);
}
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  TriangleV2.h                                                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/TriangleV2.h                */
/*-------------------------------------------------------------------------*/

#ifndef TRIANGLE_V2_HEADER
#define TRIANGLE_V2_HEADER

#include "SegListV2.h"

class TriangleV2 : public SegListV2 {
 public:
  TriangleV2() {};
  ~TriangleV2() {};

  std::string getSpec() const;
};

#endif
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  TriangleV2.cpp                                                  */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/Triangle.cpp                */
/*-------------------------------------------------------------------------*/

#include "TriangleV2.h"

using namespace std;

//---------------------------------------------------------------------------
// Procedure: getSpec()

string TriangleV2::getSpec() const
{
  string spec = "type=triangle,";
  spec += SegListV2::getSpec();
  return(spec);
}
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SquareV2.h                                                      */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SquareV2.h                  */
/*-------------------------------------------------------------------------*/

#ifndef SQUARE_V2_HEADER
#define SQUARE_V2_HEADER

#include "SegListV2.h"

class SquareV2 : public SegListV2 {
 public:
  SquareV2() {};
  ~SquareV2() {};

  std::string getSpec() const;
};
#endif
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SquareV2.cpp                                                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/TriangleV2.cpp              */
/*-------------------------------------------------------------------------*/

#include "SquareV2.h"

using namespace std;

//-------------------------------------------------------------------------
// Procedure: getSpec()

string SquareV2::getSpec() const
{
  string spec = "type=square,";
  spec += SegListV2::getSpec();
  return(spec);
}
```

4.3 Solution to Exercise 3

```
/*-------------------------------------------------------------------------*/
/*  FILE:  shapes_v3.cpp (Eighth C++ Lab Exercise 3)                       */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/shapes_v3.cpp               */
/*  RUN:   shapes_v3                                                       */
/*-------------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include "TriangleV3.h"
#include "SquareV3.h"

using namespace std;

int main()
{
  TriangleV3 triangle;
  Vertex t_vertex1(0,0);
  Vertex t_vertex2(10,0);
  Vertex t_vertex3(5,5);
  triangle.addVertex(t_vertex1);
  triangle.addVertex(t_vertex2);
  triangle.addVertex(t_vertex3);

  SquareV3 square;
  Vertex s_vertex1(0,0);
  Vertex s_vertex2(10,0);
  Vertex s_vertex3(10,10);
  Vertex s_vertex4(0,10);
  square.addVertex(s_vertex1);
  square.addVertex(s_vertex2);
  square.addVertex(s_vertex3);
  square.addVertex(s_vertex4);

  cout << "Triangle: " << triangle.getSpec() << endl;
  cout << "    Area: " << triangle.getArea() << endl;
  cout << "Square:   " << square.getSpec() << endl;
  cout << "    Area: " << square.getArea() << endl;

  return(0);
}
```

```
/*-------------------------------------------------------------------*/
/*  FILE:  SegListV3.h                                               */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV3.h            */
/*-------------------------------------------------------------------*/

#ifndef SEGLIST_V3_HEADER
#define SEGLIST_V3_HEADER

#include <string>
#include <vector>
#include "Vertex.h"

class SegListV3 {
 public:
  SegListV3() {};
  ~SegListV3() {};

  void addVertex(Vertex vertex) {m_vertices.push_back(vertex);};

  std::string getSpec() const;

  double getMinX() const;
  double getMaxX() const;
  double getMinY() const;
  double getMaxY() const;

 protected:
  std::vector<Vertex> m_vertices;
};
#endif
```

```
/*-----------------------------------------------------------------------*/
/*  FILE:  SegListV3.cpp                                                 */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV3.cpp             */
/*-----------------------------------------------------------------------*/

#include "SegListV3.h"

using namespace std;

//---------------------------------------------------------------------
// Procedure: getSpec

string SegListV3::getSpec() const
{
  string spec;
  for(unsigned int i=0; i<m_vertices.size(); i++) {
    if(i != 0)
      spec += ",";
    spec += m_vertices[i].getSpec();
  }
  return(spec);
}


//---------------------------------------------------------------------
// Procedure: getMinX()

double SegListV3::getMinX() const
{
  if(m_vertices.size() == 0)
    return(0);
  double min_x = m_vertices[0].getX();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getX() < min_x)
      min_x = m_vertices[i].getX();
  }
  return(min_x);
}

//---------------------------------------------------------------------
// Procedure: getMaxX()

double SegListV3::getMaxX() const
{
  if(m_vertices.size() == 0)
    return(0);
  double max_x = m_vertices[0].getX();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getX() > max_x)
      max_x = m_vertices[i].getX();
  }
  return(max_x);
}
```

```
//---------------------------------------------------------------------
// Procedure: getMinY()

double SegListV3::getMinY() const
{
  if(m_vertices.size() == 0)
    return(0);
  double min_y = m_vertices[0].getY();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getY() < min_y)
      min_y = m_vertices[i].getY();
  }
  return(min_y);
}

//---------------------------------------------------------------------
// Procedure: getMaxY()

double SegListV3::getMaxY() const
{
  if(m_vertices.size() == 0)
    return(0);
  double max_y = m_vertices[0].getY();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getY() > max_y)
      max_y = m_vertices[i].getY();
  }
  return(max_y);
}
```

```
/*---------------------------------------------------------------------*/
/*  FILE:  TriangleV3.h                                                */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/TriangleV3.h             */
/*---------------------------------------------------------------------*/

#ifndef TRIANGLE_V3_HEADER
#define TRIANGLE_V3_HEADER

#include <string>
#include <vector>
#include "SegListV3.h"

class TriangleV3 : public SegListV3 {
 public:
  TriangleV3() {};
  ~TriangleV3() {};

  std::string getSpec() const;

  double getArea() const;
};

#endif
```

```
/*--------------------------------------------------------------------*/
/*  FILE:  TriangleV3.cpp                                             */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/TriangleV3.cpp         */
/*--------------------------------------------------------------------*/

#include "TriangleV3.h"

using namespace std;

//--------------------------------------------------------------------
// Procedure: getSpec()

string TriangleV3::getSpec() const
{
  string spec = "type=triangle,";
  spec += SegListV3::getSpec();
  return(spec);
}

//--------------------------------------------------------------------
// Procedure: getArea()

double TriangleV3::getArea() const
{
  // First compute the area of the bounding box
  double xmin = getMinX();
  double xmax = getMaxX();
  double ymin = getMinY();
  double ymax = getMaxY();

  double square_area = (xmax-xmin) * (ymax-ymin);

  return(square_area / 2);
}
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SquareV3.h                                                      */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SquareV3.h                  */
/*-------------------------------------------------------------------------*/

#ifndef SQUARE_V3_HEADER
#define SQUARE_V3_HEADER

#include <string>
#include "SegListV3.h"

class SquareV3 : public SegListV3 {
 public:
  SquareV3() {};
  ~SquareV3() {};

  std::string getSpec() const;

  double getArea() const;
};

#endif
```

```
/*--------------------------------------------------------------------*/
/*  FILE:  SquareV3.cpp                                               */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SquareV3.cpp            */
/*--------------------------------------------------------------------*/

#include "SquareV3.h"

using namespace std;

//--------------------------------------------------------------------
// Procedure: getSpec()

string SquareV3::getSpec() const
{
  string spec = "type=square,";
  spec += SegListV3::getSpec();
  return(spec);
}

//--------------------------------------------------------------------
// Procedure: getArea()

double SquareV3::getArea() const
{
  // First compute the area of the bounding box
  double xmin = getMinX();
  double xmax = getMaxX();
  double ymin = getMinY();
  double ymax = getMaxY();

  double area = (xmax-xmin) * (ymax-ymin);

  return(area);
}
```

4.4 Solution to Exercise 4

```
/*-------------------------------------------------------------------------*/
/*  FILE:  shapes_v4.cpp (Eighth C++ Lab Exercise 4)                       */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/shapes_v4.cpp               */
/*  RUN:   shapes_v4                                                       */
/*-------------------------------------------------------------------------*/

#include <iostream>      // For use of the cout function
#include "TriangleV4.h"
#include "SquareV4.h"

using namespace std;

int main()
{
  TriangleV4 triangle;
  Vertex t_vertex1(0,0);
  Vertex t_vertex2(10,0);
  Vertex t_vertex3(5,5);
  triangle.addVertex(t_vertex1);
  triangle.addVertex(t_vertex2);
  triangle.addVertex(t_vertex3);

  SquareV4 square;
  Vertex s_vertex1(0,0);
  Vertex s_vertex2(10,0);
  Vertex s_vertex3(10,10);
  Vertex s_vertex4(0,10);
  square.addVertex(s_vertex1);
  square.addVertex(s_vertex2);
  square.addVertex(s_vertex3);
  square.addVertex(s_vertex4);

  SegListV4 *shape1 = &triangle;
  SegListV4 *shape2 = &square;

  cout << "Triangle: " << shape1->getSpec() << endl;
  cout << "    Area: " << shape1->getArea() << endl;
  cout << "Square:   " << shape2->getSpec() << endl;
  cout << "    Area: " << shape2->getArea() << endl;

  return(0);
}
```

```
/*--------------------------------------------------------------------*/
/*  FILE:  SegListV4.h                                                */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV4.h             */
/*--------------------------------------------------------------------*/

#ifndef SEGLIST_V4_HEADER
#define SEGLIST_V4_HEADER

#include <string>
#include <vector>
#include "Vertex.h"

class SegListV4 {
 public:
  SegListV4() {};
  ~SegListV4() {};

  void addVertex(Vertex vertex) {m_vertices.push_back(vertex);};

  std::string getSpec() const;

  virtual double getArea() const {return(0);};

  double getMinX() const;
  double getMaxX() const;
  double getMinY() const;
  double getMaxY() const;

 protected:
  std::vector<Vertex> m_vertices;
};
#endif
```

```
/*-------------------------------------------------------------------*/
/*  FILE:  SegListV4.cpp                                             */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV4.cpp          */
/*-------------------------------------------------------------------*/

#include "SegListV4.h"

using namespace std;

//---------------------------------------------------------------------
// Procedure: getSpec

string SegListV4::getSpec() const
{
  string spec;
  for(unsigned int i=0; i<m_vertices.size(); i++) {
    if(i != 0)
      spec += ",";
    spec += m_vertices[i].getSpec();
  }
  return(spec);
}


//---------------------------------------------------------------------
// Procedure: getMinX()

double SegListV4::getMinX() const
{
  if(m_vertices.size() == 0)
    return(0);
  double min_x = m_vertices[0].getX();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getX() < min_x)
      min_x = m_vertices[i].getX();
  }
  return(min_x);
}

//---------------------------------------------------------------------
// Procedure: getMaxX()

double SegListV4::getMaxX() const
{
  if(m_vertices.size() == 0)
    return(0);
  double max_x = m_vertices[0].getX();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getX() > max_x)
      max_x = m_vertices[i].getX();
  }
  return(max_x);
}
```

```
//---------------------------------------------------------------------
// Procedure: getMinY()

double SegListV4::getMinY() const
{
  if(m_vertices.size() == 0)
    return(0);
  double min_y = m_vertices[0].getY();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getY() < min_y)
      min_y = m_vertices[i].getY();
  }
  return(min_y);
}

//---------------------------------------------------------------------
// Procedure: getMaxY()

double SegListV4::getMaxY() const
{
  if(m_vertices.size() == 0)
    return(0);
  double max_y = m_vertices[0].getY();
  for(unsigned int i=1; i<m_vertices.size(); i++) {
    if(m_vertices[i].getY() > max_y)
      max_y = m_vertices[i].getY();
  }
  return(max_y);
}
```

```
/*---------------------------------------------------------------------*/
/*  FILE:  TriangleV4.h                                                */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/TriangleV4.h            */
/*---------------------------------------------------------------------*/

#ifndef TRIANGLE_V4_HEADER
#define TRIANGLE_V4_HEADER

#include <string>
#include <vector>
#include "SegListV4.h"

class TriangleV4 : public SegListV4 {
 public:
  TriangleV4() {};
  ~TriangleV4() {};

  std::string getSpec() const;

  double getArea() const;
};

#endif
```

```
/*--------------------------------------------------------------------*/
/*  FILE:  TriangleV4.cpp                                             */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/TriangleV4.cpp         */
/*--------------------------------------------------------------------*/

#include "TriangleV4.h"

using namespace std;

//--------------------------------------------------------------------
// Procedure: getSpec()

string TriangleV4::getSpec() const
{
  string spec = "type=triangle,";
  spec += SegListV4::getSpec();
  return(spec);
}

//--------------------------------------------------------------------
// Procedure: getArea()

double TriangleV4::getArea() const
{
  // First compute the area of the bounding box
  double xmin = getMinX();
  double xmax = getMaxX();
  double ymin = getMinY();
  double ymax = getMaxY();

  double square_area = (xmax-xmin) * (ymax-ymin);

  return(square_area / 2);
}
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SquareV4.h                                                      */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SquareV4.h                  */
/*-------------------------------------------------------------------------*/

#ifndef SQUARE_V4_HEADER
#define SQUARE_V4_HEADER

#include <string>
#include "SegListV4.h"

class SquareV4 : public SegListV4 {
 public:
  SquareV4() {};
  ~SquareV4() {};

  std::string getSpec() const;

  double getArea() const;
};
#endif
```

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SquareV4.cpp                                                    */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SquareV4.cpp                */
/*-------------------------------------------------------------------------*/

#include "SquareV4.h"

using namespace std;

//---------------------------------------------------------------------------
// Procedure: getSpec()

string SquareV4::getSpec() const
{
  string spec = "type=square,";
  spec += SegListV4::getSpec();
  return(spec);
}

//---------------------------------------------------------------------------
// Procedure: getArea()

double SquareV4::getArea() const
{
  // First compute the area of the bounding box
  double xmin = getMinX();
  double xmax = getMaxX();
  double ymin = getMinY();
  double ymax = getMaxY();

  double area = (xmax-xmin) * (ymax-ymin);

  return(area);
}
```

4.5 Solution to Exercise 5

```
/*-------------------------------------------------------------------------*/
/*  FILE:  SegListV4.h                                                     */
/*  WGET:  wget http://oceanai.mit.edu/cpplabs/SegListV4.h                 */
/*-------------------------------------------------------------------------*/

#ifndef SEGLIST_V4_HEADER
#define SEGLIST_V4_HEADER

#include <string>
#include <vector>
#include "Vertex.h"

class SegListV4 {
 public:
  SegListV4() {};
  ~SegListV4() {};

  void addVertex(Vertex vertex) {m_vertices.push_back(vertex);};

  virtual std::string getSpec() const;   // Only changed line!! now virtual

  virtual double getArea() const {return(0);};

  double getMinX() const;
  double getMaxX() const;
  double getMinY() const;
  double getMaxY() const;

 protected:
  std::vector<Vertex> m_vertices;
};
#endif
```

4.6 A Makefile for All Exercises in this Lab

```
# Makefile for Lab 08 (Ten Short C++ Labs)
# wget http://oceanai.mit.edu/cpplabs/lab08/Makefile

all: shapes_v1 shapes_v2 shapes_v3 shapes_v4

#-----------------------------------------------------------------------
# Builing a simple Triangle and Square subclass
v1_source  = shapes_v1.cpp Triangle.cpp Square.cpp
v1_headers = Triangle.h Square.h

shapes_v1: ../lib_geometry/libgeometry.a $(v1_source) $(v1_headers)
        g++ -I../lib_geometry -L../lib_geometry -lgeometry -o shapes_v1 $(v1_source)

#-----------------------------------------------------------------------
# Using an inherited function getPerimeter in the subclasses
v2_source  = shapes_v2.cpp TriangleV2.cpp SquareV2.cpp SegListV2.cpp
v2_headers = TriangleV2.h SquareV2.h SegListV2.h

shapes_v2: ../lib_geometry/libgeometry.a $(v2_source) $(v2_headers)
        g++ -I../lib_geometry -L../lib_geometry -lgeometry -o shapes_v2 $(v2_source)

#-----------------------------------------------------------------------
# Polymorphism: implementing an area() function in the subclasses
v3_source  = shapes_v3.cpp TriangleV3.cpp SquareV3.cpp SegListV3.cpp
v3_headers = TriangleV3.h SquareV3.h SegListV3.h

shapes_v3: ../lib_geometry/libgeometry.a $(v3_source) $(v3_headers)
        g++ -I../lib_geometry -L../lib_geometry -lgeometry -o shapes_v3 $(v3_source)

#-----------------------------------------------------------------------
# Virtual functions: making area() virtual, using base-class pointers
v4_source  = shapes_v4.cpp TriangleV4.cpp SquareV4.cpp SegListV4.cpp
v4_headers = TriangleV4.h SquareV4.h SegListV4.h

shapes_v4: ../lib_geometry/libgeometry.a $(v4_source) $(v4_headers)
        g++ -I../lib_geometry -L../lib_geometry -lgeometry -o shapes_v4 $(v4_source)

#-----------------------------------------------------------------------
clean:
        rm -f shapes_v1 shapes_v2 shapes_v3 shapes_v4
```