Name Anders Dahl **EECE 2160** **Spring 2016**

Introduction to C++

The purpose of this exercise is to:

- Illustrate the structure of C++ classes
- Illustrate functions declaration and definition inside and outside the class body
- Illustrate the use of constructors
- Illustrate calling methods within a class

STEP 1 – Define class member functions

1. We are going to start with a C++ program with the following UML diagram for the Rectangle class:

Rectangle

```
- width : int
- length : int
```

```
+ setWidth(int w) void
+ setLenght() void
+ getWidth() int
+ getLenght() int
+ getAarea() int
+ grow(int factor) void
```

2. Log in to your COE account and copy or import the following C++ program from Blackboard under "Lecture Notes".

```
#include <iostream>

using namespace std;

class Rectangle
{
private:
   int width;
   int length;
public:
   void setWidth(int w);
   void setLength(int l);
   int getWidth() { return width; }
   int getLength() { return length; }
   int getArea();
   void grow(int factor);
};
void Rectangle::setWidth(int w)
{
    width = w;
}
void Rectangle::setLength(int l)
{
    length = l;
}
 // Put the function definitions here
```

```
    int main()
    {


       cout << myRect.getWidth() << " * " << myRect.getLength() << " = ";
       cout <<  myRect.getArea() << endl;

       myRect.grow(3);
       cout << myRect.getWidth() << " * " << myRect.getLength() << " = ";
       cout <<  myRect.getArea() << endl;
}
```

3. All attributes and operations in a class (data members and functions) are designated as either private, protected, or public with the following meanings:
   - public
     - may be accessible from anywhere within a program
   - private
     - may be accessed only by the member functions, and friends of this class, not open for nonmember functions
   - protected
     - acts as public for derived classes (virtual)
     - behaves as private for the rest of the program

   In the Rectangle class, the data members `width` and `length` are only accessible within the Rectangle class, and can be operated by the functions of that class only. **Note:** `main` function is not a member of the Rectangle class.

4. Class member functions are always declared inside the class body the same way you would declare a function prototype.
5. Their definitions can be placed inside the class body, or outside the class body. In the class Rectangle, the functions `getWidth()` and `getLength()` are defined inside the class body. If a function is defined outside the class body, you have to indicate that they belong to the class by including the class' name and scope operator (`::`) in the declaration line. The functions `setWidth()` and `setLength()` are defined outside the class, but the use of the class name and scope operator (`Rectangle::`) indicate they belong to the 2eRectangle class.

   The function `getArea()` declared in the class body has the following definition:
   ```
   {
         return width * length;
   }
   ```
   Put this definition <u>inside</u> the class body.

   The function `grow` () declared in the class body has the following definition:
   ```
   {
         width = width * factor;
         length = length * factor;
   }
   ```
   Put this definition <u>outside</u> the class body. Don't forget to include the return type, class name, and scope operator.

6. Compile the Program with the `g++` compiler and run it. Note and explain the output in the form on the last page.
7. Uncomment the two line in `main` function. Compile the program again noting and explaining the output.

<u>STEP 2 – Add some constructors</u>

1. Within a class, you can provide special functions that are automatically invoked when you create a **new** object. These special methods are called **constructors.** Constructors have some special coding guidelines in that they:

- Have the same name as the class.
- Their definition does not need a return type (not even void). This is because they return an object of the class type, by definition.
- They go in the public part of the class
- They are invoked automatically without explicit function call

The purpose of constructor functions? They can be used to **initialize instance variables during construction** of an object.

Here is an example constructor. Add this to the Rectangle class, outside the class body. Include the prototype inside the class body:

```
/**
 * @param w width
 * @param l length
 */
Rectangle::Rectangle(int w, int l)  // notice no return type is needed!
{
      width = w;
      length = l;
}
```

Test this constructor by making your `main` look like this (You can comment out the unused lines of code):

```
int main() {
      Rectangle myRect(6, 8);
      cout << myRect.getWidth() << " * " << myRect.getLength() << " = ";
      cout <<  myRect.getArea() << endl;

      myRect.grow(3);
      cout << myRect.getWidth() << " * " << myRect.getLength() << " = ";
      cout <<  myRect.getArea() << endl;
}
```

Notice the **parameters** in the constructor: width = 6 and length = 8! Test and record the output of the revised program in the form at the end.

2. Once you add a constructor, you should also provide a **default constructor** that has no parameters. Add this default constructor to the Rectangle class inside the class body:

```
/**
 * Default Constructor
 */
Rectangle()
{
      width = length = 5;
}
```

Notice that it **initializes the instance** variables to a "default" value.
Test the default constructor by adding the following lines to your main function:

```
Rectangle myRect2; // note that no values are passed for width & length
cout << myRect2.getWidth() << " * " << myRect2.getLength() << " = ";
cout <<  myRect2.getArea() << endl;
```

Compare the values of the two objects, `myRect` and **`myRect2`**. Record your output in the form below.

STEP 3 – Using a method of the class from within the class

Let’s add another function `setValues()` to our Rectangle class. The function receives, as arguments, two integers; width and length, and sets the data members to those values. The function looks like this:

```
/**
 * @param w width
 * @param l length
 */
void Rectangle::setValues(int w, int l)    {
      width = w;
      length = l;
}
```

The definition of this function looks similar to the definition of one of our constructors. One reason to have another function that does the same thing is that we can call it at any time we want to update both data members at the same time without creating another object. In fact, we can modify the constructor to use this function to initialize the data members. Modify the constructor to look like this:

```
/**
 * @param w width
 * @param l length
 */
Rectangle::Rectangle(int w, int l)  // notice no return type is needed!
{
      setValues(w, l);
}
```

Notice that we are calling the `setValues()` function from the constructor. Test the program and confirm that it is doing the same thing as before. Record your output in the form below.

STEP 4 – Creating multiple objects of a class

To create multiple objects of the Rectangle class, you can create an array of objects in the`main` function with the following line;

```
Rectangle manyRects[5]; // Create an array of 5 objects
```

Using the `setValues()` function, set the following dimensions for 5 rectangles, and print out their areas using the `getArea()` function.

(width, length) = (3, 2), (6,4), (9, 8), (12, 16), (15, 32)

Note: *You can comment out the unused lines of code in your main function. Do not delete them.*

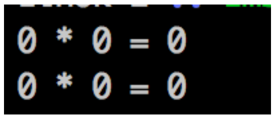
Show your output in the form below.

Name: Anders Dahl

Spring 2016 **EECE2160** **J. Kimani**

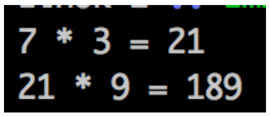
C++ Exercise 1

1. Ouptut from Step 1:



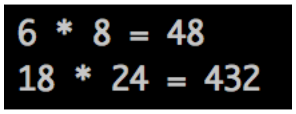
```
0 * 0 = 0
0 * 0 = 0
```

The rectangle does not have a width or a height – they were never set.
After uncommenting the code, the rectangle does have a width and a height set.



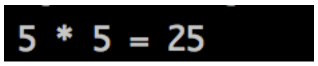
```
7 * 3 = 21
21 * 9 = 189
```

2. Output from Step 2 (Constructors):
   a. With the first constructor?



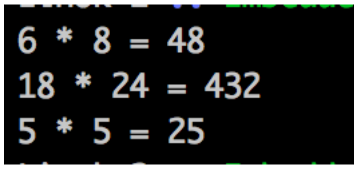
```
6 * 8 = 48
18 * 24 = 432
```

   b. With default constructor?



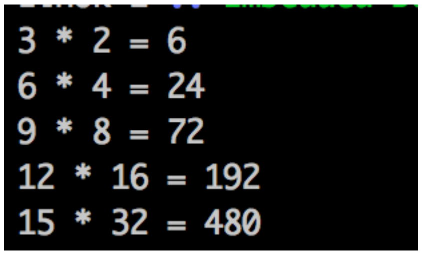
```
5 * 5 = 25
```

3. Output from Step 3:



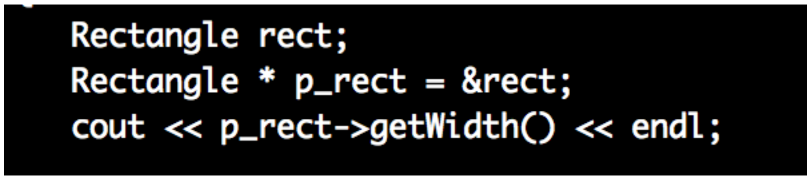
```
6 * 8 = 48
18 * 24 = 432
5 * 5 = 25
```

4. Output from Step 4 (Array of objects):

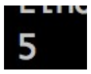


```
3 * 2 = 6
6 * 4 = 24
9 * 8 = 72
12 * 16 = 192
15 * 32 = 480
```

5. Show how you would create a pointer object of the class `Rectangle`, and how you would use it to access the `width` member data from the `main` function. Test this in your program.



```
Rectangle rect;
Rectangle * p_rect = &rect;
cout << p_rect->getWidth() << endl;
```



```
5
```

6. Attach your final C++ program here (you can copy and paste it):

```
#include <iostream>

using namespace std;

class Rectangle
{
private:
   int width;
         int length;
public:
   /**
    * Default Constructor
    */
   Rectangle()
   {
      width = length = 5;
   }
   Rectangle(int w, int l);
   void setValues(int w, int l);
   void setWidth(int w);
   void setLength(int l);
         int getWidth() { return width; }
         int getLength() { return length; }
         int getArea() { return width * length; };
         void grow(int factor);
};
/**
 * @param w width
 * @param l length
 */
Rectangle::Rectangle(int w, int l)  // notice no return type is needed!
{
   setValues(w, l);
}
/**
 * @param w width
 * @param l length
 */
void Rectangle::setValues(int w, int l) {
   width = w;
   length = l;
}
void Rectangle::setWidth(int w)
{
   width = w;
}
void Rectangle::setLength(int l)
{
   length = l;
}
void Rectangle::grow(int factor)
```

```
{
    width = width * factor;
    length = length * factor;
}
int main()
{
   Rectangle rect;
   Rectangle * p_rect = &rect;
   cout << p_rect->getWidth() << endl;

   // Rectangle manyRects[5]; // Create an array of 5 objects
   // manyRects[0].setValues(3,2);
   // manyRects[1].setValues(6,4);
   // manyRects[2].setValues(9,8);
   // manyRects[3].setValues(12,16);
   // manyRects[4].setValues(15,32);

   // cout << manyRects[0].getWidth() << " * " << manyRects[0].getLength() << " = ";
   // cout <<  manyRects[0].getArea() << endl;

   // cout << manyRects[1].getWidth() << " * " << manyRects[1].getLength() << " = ";
   // cout <<  manyRects[1].getArea() << endl;

   // cout << manyRects[2].getWidth() << " * " << manyRects[2].getLength() << " = ";
   // cout <<  manyRects[2].getArea() << endl;

   // cout << manyRects[3].getWidth() << " * " << manyRects[3].getLength() << " = ";
   // cout <<  manyRects[3].getArea() << endl;

   // cout << manyRects[4].getWidth() << " * " << manyRects[4].getLength() << " = ";
   // cout <<  manyRects[4].getArea() << endl;

   // Rectangle myRect(6, 8);
   // cout << myRect.getWidth() << " * " << myRect.getLength() << " = ";
   // cout <<  myRect.getArea() << endl;

   // myRect.grow(3);
   // cout << myRect.getWidth() << " * " << myRect.getLength() << " = ";
   // cout <<  myRect.getArea() << endl;

   // Rectangle myRect2; // note that no values are passed for width & length
   // cout << myRect2.getWidth() << " * " << myRect2.getLength() << " = ";
   // cout <<  myRect2.getArea() << endl;

}// Put the function definitions here
```