CSE-111 Introduction to Computer Science II, Spring Term 2017
Programming Lab #1
Section 500
Due Date: 03/08/2017 at 11:59PM

Purpose:

By completing this lab you will gain experience with object-oriented programming (OOP) and re-using code.

Sections

1 Background on Classes & Objects

In our previous assignment we were introduced to enumerated data types (enum) and structured data types (structs). In this assignment we will re-use our previous code with the exception of the Person struct we created, we will replace it with a an Person class. You may ask, what is the point? Well the reason why we would want to do such a thing is because a struct's data is public and anyone can access it and change it. Classes are very similar to structs, but one of the key advantages classes have over structs is **by default a class's data is private** and the objects made from classes are responsible for maintaining themselves. Since an object's data is not readily avaialable to anyone to access and change, the data is more secure and less prone to bugs.

Classes are basically blueprints for how to build copies of a specific type of object. More formally put, a class is code that specifies the **attributes** and **member functions** that a particular type of object may have. A class is not an object, but it is a description of an object. However, classes are used to make objects. Each object that is created from a class is called an instance of the class. Welcome to the world of object oriented programming, where classes build objects, and objects allow us to create digital worlds.

1.1 Seperating Class Specification from Implementation

Classes can be broken down into two seperate files. A specification file or **header or .h file** and an **implementation or .cpp file**. The header file only lists the attributes and member functions of the class. Attributes are the associated pieces of data that the class holds. The member functions are the actions that the class can perform. The public interface are member functions with an access level of public, which allows a program to use the dot operator to call member functions to access or alter an objects hidden data. Inside of the header file all of the access levels are established for all the attributes and member functions for the class. Access levels will be covered in the access specifier section of this lab.

1.2 header file Syntax

```
#ifndef COMPLEX_H //<-- directive to see if our class exist
#define COMPLEX_H //<-- define our class if non-existing

#include <iostream> // <-- cout
using namespace std; // <-- string and endl

class Complex // <-- name of the class
{
    private: // <-- access specifier
        // --- private attributes listed below ---
        float real;
        float imaginary;
        // --- private attributes listed above ---
    public: // <-- access specifier
        // --- public member functions listed below ---
        Complex(float r, float i); // <-- constructor
        ~Complex(); // destructor
        float getReal() const;  // <-- accessor method
        float getImaginary() const; // <-- accessor method
        void setReal(float r);  // <-- mutator method
        void setImaginary(float i);
        Complex add(Complex c) const;
        // --- public member functions listed above ---
};
#endif // <-- end the if NOT defined statement
```

1.3 Exercise #1

1. Create a Person.h header file
2. include the **"using namespace std;**" near the top of the header file so the Person class can have string objects as attributes
3. Then cut and paste your Gender and Status definitions from your previous assignment into the Person.h header file. Paste this code after the using statement
4. After the enum definitions, Defined our Person class with the following attributes and member functions. The attributes are the same as our Person struct.
   (a) string firstName (attribute)

(b) string lastName (attribute)

(c) Gender gender (attribute)

(d) Status status (attribute)

(e) Person* spouse (attribute)

(f) Person(string firstName, string lastName, Gender gender, Status status, Person* spouse); // constructor

(g) string getFirstName() const;

(h) string getLastName() const;

(i) string personToString() const;

(j) Gender getGender() const;

(k) Status getStatus() const;

(l) void setFirstName(string firstName);

(m) void setlastName(string lastName);

(n) void setStatus(Status s);

1.4 Access Specifiers

By default anything defined within a class is marked as private. You need to control your class's access levels inorder for your program to work with your class and retrieve the data stored within your class. Access specifiers control who can access attributes or functions of your objects. The two access specifiers we covered in lecture are **public** and **private**. Once an access specifier is declared in your code all lines of code that follow it has that level until another access specifier is declared. Review the code in the header file syntax section to see how to set the access specifiers.

1.5 Exercise #2

Update the Person.h file to include access specifiers.

- Mark ALL of the attributes as private (this will hide the data)
- Mark ALL of the member functions as public (this will allow us to access the data in our main code, by using the public member functions)

2 Class Implementation

Now that we have finished the specification file for the Person class we need to implement all of the functions. The implementation is the actual code that will run when different member functions are called. Only when the Class Implementation file is complete will you objects be able to come to life.

For each function defined within the header file you must provided implementation unless you provide inline implementation within the header file. In order to implement a function you need to know 4 pieces of information; the return type, the name of the class, the name of the function and the parameter list

2.1 implementation file Syntax

```
#include "Complex.h" // <-- must link to the header file

// BELOW IMPLEMENT EACH FUNCTION DEFINED IN THE HEADER

Complex::Complex(float real, float imaginary)
{
    this->real = real;
    this->imaginary = imaginary;
    cout << "constructor was called" << endl;
}

Complex::~Complex()
{
    cout << "destructor was called" << endl;
}

float Complex::getReal() const{

    return this->real;
}

float Complex::getImaginary() const{
    return this->imaginary;
}

void Complex::setReal(float real)
{
    cout << "setReal() was called" << endl;
    this->real = real;
}

void Complex::setImaginary(float imaginary)
{
    cout << "setImaginary() was called" << endl;
    this->imaginary = imaginary;
}


Complex Complex::add(Complex c) const
{
    float r = this->real + c.getReal();
    float i = this->imaginary + c.getImaginary();
    return Complex(r,i);
}
```

2.2 A few notes:

- **::** - is the scope resolution operator, it helps to identify and specify the context to which an identifier refers
- Complex::getReal() - refers to the getReal() function declared in the class Complex
- when implementing the functions of the class the scope resolution operator MUST be used.

2.3 Special functions with the same name as the class

1. Constructor - is a member function that is automatically called when an object is created.
   (a) Purpose - **initialize** the object's **data**
   (b) if **no constructor** is specified, a **default** constructor will **automatically be created** that does nothing

   objects of the class
2. Destructor - is a member function that is automatically called when an object
   (a) Purpose - run a **shutdown** rountine when the object goes out of existence, a common example is deallocating space if neccessary.
   (b) if **no destructor** is specified, a **default** destructor will **automatically be created** that does nothing

2.4 OOP Syntax

```
#include "Complex.h" // <-- bring the Complex class into scope
using namespace std;

int main()
{
    // initialize 3 Complex objects
    Complex number1(5.0,2.0); // call the constructor

     // dynamically allocate a Complex object
    Complex* number2Ptr = new Complex(-5.0,2.0);

    // call the add member function
    /* Note: because number2Ptr is a pointer and add is
       looking for a Complex object we dereference number2Ptr
       inorder to call the add function with it */
    Complex number3 = number1.add(*number2Ptr);


    // display the contents of the object
    // with their get functions
    cout << "number1 = " << number1.getReal()  << " + "
         << number1.getImaginary() << "j" << endl;
    cout << "number2 = " << number2Ptr->getReal()  << " + "
         << number2Ptr->getImaginary() << "j" << endl;
    cout << "number3 = " << number3.getReal()  << " + "
         << number3.getImaginary() << "j" << endl;

    number1.setReal(100.0);
    number1.setImaginary(100.0);
    cout << "number1 = " << number1.getReal()  << " + "
         << number1.getImaginary() << "j" << endl;

    delete number2Ptr;
    // when the program concludes the destructors
    // for the number1 and number3 gets called
    return 0;
}
```

2.5 Exercise #3

Implement ALL of the functions specified in the Person.h file.

1. Person(string firstName, string lastName, Gender gender, Status status, Person* spouse); // constructor
   - this constructor should **intialize** firstName, lastName, gender, status to SINGLE and spouse to NULL
2. string getFirstName() const;
   - this function returns the firstName attribute
3. string getLastName() const;
   - this function returns the lastName attribute
4. string toString() const;
   - this function returns the same string that the personToString function from the previous assignment returned
5. Gender getGender() const;
   - this function returns the gender attribute
6. Status getStatus() const;
   - this function returns the status attribute
7. void setFirstName(string firstName);
   - this function sets firstName attribute

8. void setlastName(string lastName);
   - this function sets the lastName attribute
9. void setStatus(Status s);
   - this function sets the status attribute

2.6 Refactor the code

Now that we have re-designed our Person data type we need to include our Person.h file in our main code and modify our main code to work with our person objects.

2.7 Exercise #3

Update the main program to use our new Person data type

1. Type **#include "Person.h"** in the section where we include header files, this will bring the Person class into scope for the main to use
2. **Delete** the Gender, Status and Person data types defined in the main file. All of this is now defined inside the Person.h file
3. **Delete** the personToString() and instead use the toString() member function defined in the Person class.
4. everywhere Person attributes were directly accessed with the dot operator, you need to use the dot operator with the appropriate member function from exercise 3 to get or set the object's data
5. you need to use a constructor to initialize the data for each of the Person objects in the "people" array.
6. If you do all this you should be able to generate the same results from the previous assignment.