PS # 6

Deadline: 01.05.2020 at 23:50

Hand-in Policy:

Create a cpp file for each part of the assignment and save the screenshot while these parts are running.

Your assignment should consist of 3 cpp files and 3 screenshots.

Zip all files of the assignment. Your zip files should look like this

Ps6_studentnumber_name_surname.zip

1)

Define a class which name is Complex for complex numbers. A complex number is $a + b*i$ where for our purposes, a and b are numbers of type double, and i is a number that represents the quantity $\sqrt{-1}$. Represent a complex number as two values of type double. Name the member variables real and maginary. Include a constructor with two parameters of type double that can be used to set the member variables of an object to any values. Include a constructor that has only a single parameter of type double; call this parameter realPart and define the constructor so that the object will be initialized to realPart + 0*i . Include a default constructor that initializes an object to 0 (that is, to 0 + 0*i). Overload all the following operators so that they correctly apply to the type Complex :

== , + , - , * , >> , and << .

You should also write a test program to test your class. *Hints*: To add or subtract two complex numbers, add or subtract the two member variables of type double . The product of two complex numbers is given by the following formula:

$(a + b*i)*(c + d*i) = = (a*c - b*d) + (a*d + b*c)*i$

In the interface file, you should define a constant i as follows:

```
const Complex i(0, 1);
```

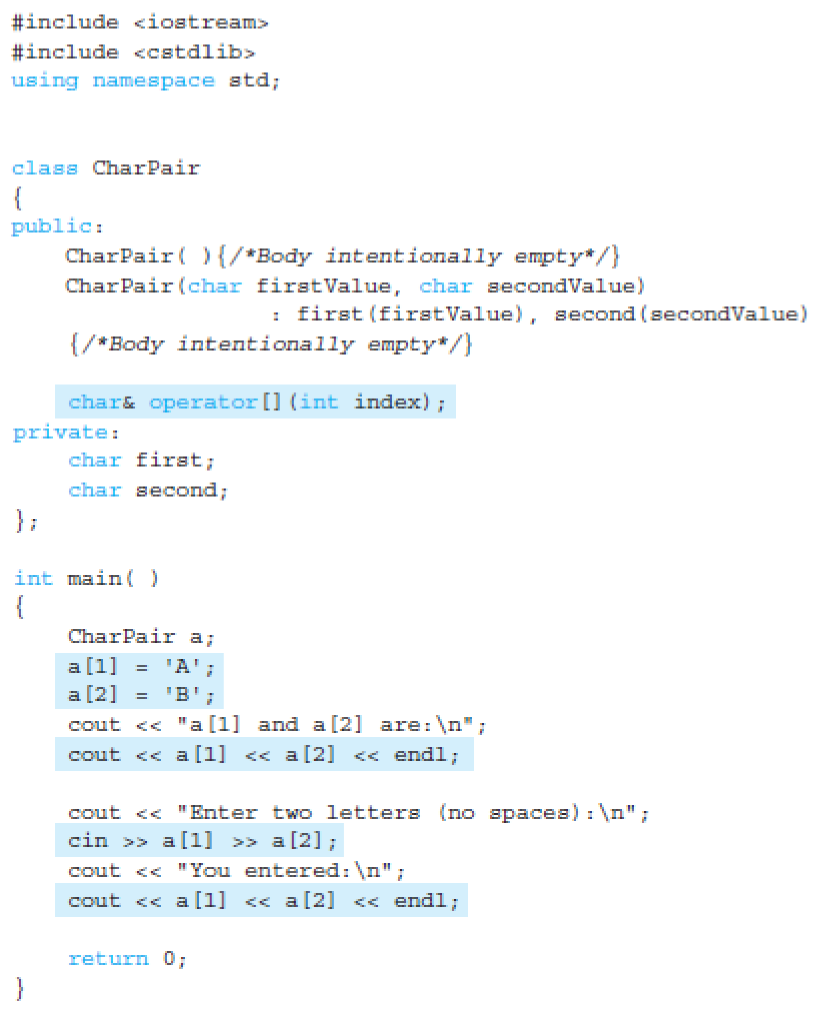
2)

Cumulatively modify the example belown as follows.

a. In Display 8.7 , replace the private char members first and second with an array of char of size 100 and a private data member named size . Provide a default constructor that initializes size to 10 and sets the first 10 of the char positions to '#'. (This only uses 10 of the possible 100 slots.) Provide an accessor function that returns the value of the private member size .
   Test.

b. Add an operator[] member that returns a char& that allows the user to Access or to set any member of the private data array using a non-negative index that is less than size .
   Test.
c. Add a constructor that takes an int argument, sz , that sets the first sz members of the char array to '#' .
   Test.
d. Add a constructor that takes an int argument, sz , and an array of char of size sz . The constructor should set the first sz members of the private data array to the sz members of the argument array of char .
   Test.

NOTES: When you test, you should test with known good values, with data on boundaries and with deliberately bad values. You are not required to put checks for index out of bounds errors in your code, but that would be a nice touch. Error handling alternatives: Issue an error message then die (that is, call exit(1)) or give the user another chance to make a correct entry.



```
#include <iostream>
#include <cstdlib>
using namespace std;


class CharPair
{
public:
    CharPair( ){/*Body intentionally empty*/}
    CharPair(char firstValue, char secondValue)
                   : first(firstValue), second(secondValue)
    {/*Body intentionally empty*/}

    char& operator[](int index);
private:
    char first;
    char second;
};

int main( )
{
    CharPair a;
    a[1] = 'A';
    a[2] = 'B';
    cout << "a[1] and a[2] are:\n";
    cout << a[1] << a[2] << endl;

    cout << "Enter two letters (no spaces):\n";
    cin >> a[1] >> a[2];
    cout << "You entered:\n";
    cout << a[1] << a[2] << endl;

    return 0;
}
```

```
//Uses iostream and cstdlib:
char& CharPair::operator[](int index)
{
    if (index == 1)
        return first;
    else if (index == 2)
        return second;
    else
    {

        cout << "Illegal index value.\n";
        exit(1);
    }
}
```

3)

Define a class which name is Rational for rational numbers. A rational number is a number that can be represented as the quotient of two integers. For example, 1/2, 3/4, 64/2, and so forth are all rational numbers. (By 1/2 and so on we mean the everyday fraction, not the integer division this expression would produce in a C++ program.) Represent rational numbers as two values of type int , one for the numerator and one for the denominator. Include a constructor with two arguments that can be used to set the member variables of an object to any legitimate values. Also include a constructor that has only a single parameter of type int ; call this single parameter wholeNumber and define the constructor so that the object will be initialized to the rational number wholeNumber /1. Include a default constructor that initializes an object to 0 (that is, to 0/1). Overload the input and output operators >> and << . Numbers are to be input and output in the form 1/2 , 15/32 , 300/401 , and so forth. Note that the numerator, the denominator, or both may contain a minus sign, so -1/2 , 15/-32 , and -300/-401 are also possible inputs. Overload all the following operators so that they correctly apply to the type Rational : == , < , <= , > , >= , + , - , * , and / . Write a test program to test your class.