Assignment 3

Assignment Instructions:

- Complete the following programming exercises in C/C++.
- Questions 8 and 9 are each worth **15 marks**. All other questions are worth **10 marks**. (100 marks total)
- Aim to make your code as concise and logical as possible.
- For convenience, you can write all functions and classes in a single file.
- Write functions to be self-contained units of code, i.e. the code inside your functions must **not** rely on variables you define outside the function (i.e. no *global variables*).
- Classes can be written as one block (i.e. there is no need to separate method declarations from their definitions).
- It's recommended that you test your code by calling your functions from your main() method, but no code written in main() will be graded.
- Do not use any library functions, unless the question specifies that you can.
- Submit your answers as a ***single text .TXT document***, *no zip or Word files etc.*, with **each answer clearly numbered and your code properly formatted/indented** (use Ctrl+A then Ctrl+I in Code::Blocks to properly format your code after you are finished writing it).

This assignment is due 6 pm Monday 2nd March. No late submissions accepted.

Q1. Write a function

```
string repeatString(const string & str, unsigned int n) { . . . }
```

which takes a C++ string str and returns the string repeated n times. Test your function by using the following test code in main():

```
cout << repeatString("echo! ",5) << endl;
```

which should print out

```
echo! echo! echo! echo! echo!
```

Q2. Write a function

```
void printBigX(int N) { . . . }
```

Which uses **a for loop nested inside another for loop** to print out a 'big X' of size N x N characters. Examples of the required output are shown below.

Calling `printBigX(7)` should print:

```
X     X
 X   X
  X X
   X
  X X
 X   X
X     X
```

Calling `printBigX(12)` should print:

```
X          X
 X        X
  X      X
   X    X
    X  X
     XX
     XX
    X  X
   X    X
  X      X
 X        X
X          X
```

Q3. Write a C++ class called `Book`. A book has an **author**, a **title** and a **year** which are stored as private data members. Use C++ strings to store these members. Book should have a constructor to initialize the three values when creating a Book object. It should also have three public methods `getAuthor()`, `getTitle()` and `getYear()` which return the current value of the author, title and year respectively.

Q4. Write a function (not a method of a class)

```
int countBooksByAuthor(Book bookList[], int size, string author) { ... }
```

which takes **an array of `Book` objects** (your class from Q3) and counts the number of books by a given author. The search should count partial matches of an author's name. E.g. if searching on "Stroustrup", books by "B. Stroustrup" or "Borjn Stroustrup" should be counted. You may make use of the find() method of the C++ string class. Test that your function operates correctly using this test code in main():

```
Book books[] = {
     Book("B. Stroustrup", "A Tour of C++", "2018"),
     Book("J. Bloch", "Effective Java", "2018"),
     Book("B. Stroustrup", "The C++ Programming Language(Fourth Edition)", "2013"),
     Book("B. McLaughlin, G. Pollice, D. West", "Head First Object Oriented Analysis and Design", "2006"),
     Book("B. McLaughlin", "Java and XML", "2006") };
string searchAuthor = "McLaughlin";
cout << "There are " << countBooksByAuthor(books, 5, searchAuthor) << " books by author " << searchAuthor << endl;
```

Q5. Write a C++ class called `Complex` which represents a complex number. It should have two private data members to store the real and imaginary part of the number, a constructor that takes two values to initialize the complex number and a `print()` method which prints out the complex number in the form "a + bi". Test your class from main() using the following code:

```
Complex c1(2.7,-5.2);
c1.print();  // should print "2.7 - 5.2i"
```

Q6. Add the following two methods as member functions of your `Complex` class

```
Complex conj() { ... }
```

which returns the *complex conjugate* of the complex number as another `Complex` object, and

```
Complex multiplyBy(const Complex & z) { ... }
```

which returns the complex number multiplied by a given complex number z. Your class must function correctly when the following test code is run from main():

```
Complex c1(2.7,-5.2);
c1.print();   // prints "2.7 - 5.2i"
Complex c2 = c1.conj();
c2.print();   // prints "2.7 + 5.2i"
Complex c3 = c1.multiplyBy(c2);
c3.print();   // prints "34.33 + 0i"
```

Q7. Write a class called `Point` that represents a 2-dimensional point. Its X and Y coordinates should be stored as private data members. It should have a default constructor and a constructor taking two parameters to initialize the point's coordinates with given X and Y values. It should also have a public method of the following form:

```
double distanceTo(const Point & p)
```

which returns the distance from this point to another point p. Your class must operate correctly with this test code (when called from main())

```
Point p1(1.0,1.0);
Point p2(2.0,2.0);
cout << "Distance between points is " << p1.distanceTo(p2) << endl;
```

which should print the distance as 1.41421

Q8. Write a `Polygon` class which represents a polygon consisting of a sequence of 2D points. A polygon can have up to a maximum of 10 points. The class should store points as **an array of Point** where Point is your class of Q7. Your polygon class should have a default constructor and the following public member functions:

```
bool add(Point p) { ... }
```

which adds a new point to the polygon. This method returns false if the polygon already has 10 points when the method is called, returns true otherwise.

```
double length() { ... }
```

which calculates and returns the perimeter length of the polygon. The points stored in the polygon are assumed to be in order around its perimeter. Your class must operate correctly when used from main() as follows:

```
Polygon poly;
poly.add(Point(0,0));
poly.add(Point(1,0));
poly.add(Point(1,1));
poly.add(Point(0,1));
cout << poly.length() << endl;  // should return length of 4
```

Q9. Write a class `CyclicString` which stores a C++ string whose characters can be retrieved in sequence by successively calling a `next()` method on the object. When the end of the string is reached, a call to `next()` will start over at the being of of the string. Your class must operate correctly with the following test code in main():

```
CyclicString s("Repeat Me!");
for (int i=0; i<50; i++)
    cout << s.next();
```

This should print out `Repeat Me!Repeat Me!Repeat Me!Repeat Me!Repeat Me!`

Submission status

Submission status	Submitted for grading
Grading status	Graded
Due date	Monday, 2 March 2020, 6:00 PM
Time remaining	Assignment was submitted 27 secs late
Grading criteria	This is the assignment marking scheme for EE219 assignments.

Question 1:	Solution is missing or no reasonable attempt. ***0 points***	An attempt has been made along the right lines but the functionality and code output are not correct. ***2.5 points***	Solution is almost correct but there are obvious flaws, some requirements stated in question are not handled or function/class has wrong form. ***5 points***	Solution almost perfect, function/class has correct form and functionality, but the code is not as efficient as it could be. ***7.5 points***	Solution perfectly correct and code efficiently implemented. ***10 points***
Question 2:	Solution is missing or no reasonable attempt. ***0 points***	An attempt has been made along the right lines but the functionality and code output are not correct. ***2.5 points***	Solution is almost correct but there are obvious flaws, some requirements stated in question are not handled or function/class has wrong form. ***5 points***	Solution almost perfect, function/class has correct form and functionality, but the code is not as efficient as it could be. ***7.5 points***	Solution perfectly correct and code efficiently implemented. ***10 points***