Programming Assignment 1 (PA1)
Out: September 21, 2021, Tuesday -- DUE: October 12, 2021, Tuesday, 11:59pm
EC327 Introduction to Software Engineering – Fall 2021

Total: 100 points

- *You may use any development environment you wish, as long as it is ANSI C++ compatible. Please make sure your code compiles and runs properly under the Linux environment on the PHO307/305 (or eng-grid) machines before submitting.*
- *PAs may be submitted up to a week late at the cost of a **30% fixed penalty** (e.g., submitting a day late and a week late is equivalent). It is in your best interest to complete as many PA questions as possible before the deadline. If you have missing questions in your original submission, you may complete and submit the missing solutions during the following week. Any submissions after the deadline will be subject to the 30% penalty. No credit will be given to solutions submitted after the 1-week late submission period following the deadline.*
- *Please make sure to follow the assignment submission guidelines for **GitHub**. These will be posted in a separate document.*

Please write C++ code for solving the following problems related to data types, operations, if statements, simple loops, and basic functions (Chapters 2-3-4-5-6 of the textbook by Forouzan).

Q1: *Shortest Distance from a Point to a Line* [20 points]
Write a program that reads three x, y coordinates from the console (STDIN). The first two coordinates will be used to compute an equation of a straight line passing through each coordinate. The program should then calculate the shortest distance from the third coordinate to the line and print the result to the console (STDOUT). You may use the internet to learn the equations but NOT get code. **We will be checking for this.**

Text in blue in the examples demonstrates the user inputs entered via the keyboard.

Sample run:
```
Enter three x,y coordinates:
1 2 2 3 2 4<enter>
The shortest distance for (2,4) to the line composed of (1,2) and (2,3) is ?**. [Program Exits]
```
**You should display at least two fractional digits. You can display more of the fractional part if you like.*
***Your program needs to replace ? with the answer. ☺*

Q2: *Rock, Paper, Scissors* [20 points]

Write a program to play "rock, paper, scissors". In this game, you first prompt the user to choose "rock, paper, or scissors". After this, the computer should randomly generate one of these three. Then you should display the choice of the computer and determine the winner (rock beats scissors, scissors beat paper, paper beats rock).

Text in blue in the examples demonstrates the user inputs entered via the keyboard.

Sample run:

```
Choose Rock (0), Paper (1), or Scissors (2): <1> <enter>
Computer chooses: Rock
You win!
[Program Exits]

Choose Rock (0), Paper (1), or Scissors (2): <2> <enter>
Computer chooses: Rock
You lose!
[Program Exits]

Choose Rock (0), Paper (1), or Scissors (2): <0> <enter>
Computer chooses: Rock
You tie!
[Program Exits]
```

**Hint: http://www.cplusplus.com/reference/cstdlib/rand/*

Q3. *Hamming Distance* [25 points]

In information theory, the **Hamming distance** between two sequences of equal length is the number of positions at which the corresponding symbols are different. Put another way, it measures the minimum number of *substitutions* required to change one string into the other, or the number of *errors* that transform one string into the other.

Examples:

The Hamming distance between:

- "**toned**" and "**roses**" is 3.
- **1011101** and **1001001** is 2.
- **2173896** and **2233796** is 3.

(Description above is from Wikipedia: http://en.wikipedia.org/wiki/Hamming_distance)

Write a program that prompts the user to enter two positive integers (in decimal; up to 32-bits of precision) and computes the Hamming distance between the two numbers when the numbers are represented in base-16 (hex format). The program then displays the Hamming distance on the screen.

Hint: You don't need to convert the numbers to 1's, 2's, and 0's etc. to figure this out. Think about how to figure this out with arithmetic. Start out thinking how you would do this with base-10.

```
Sample runs:
Enter two positive integers:  <145363><enter>
<54637823><enter>
Hamming distance between 145363 and 54637823 when numbers are in hex format is: 7   [Program
Exits]*

Enter two positive integers:  <262178770><enter>
<262637534><enter>
Hamming distance between 262178770 and 262637534 when numbers are in hex format is 2.
[Program Exits]**
```

** 145363 and 54637823 are represented as 237D3 and 341B4FF, respectively, in hex form.*
***262178770 and 262637534 are represented as FA087D2 and FA787DE in hex form.*

Q4. *Palindromic Number Strings* [10 points]

A **palindromic number** (also known as a **numeral palindrome** or a **numeric palindrome**) is a number that remains the same when its digits are reversed. Like 16461 or 123321.

Write a program that prompts the user to enter a number string and determines whether or not the number is a palindrome. The number entered by the user should only contain digits 0-9 and should return false if any invalid characters are entered.

Text in blue in the examples demonstrates the user inputs entered via the keyboard.

```
Sample runs:
Enter a number:  <81818> <enter>
The number is 81818 is a palindrome. [Program Exits]

Enter a number:  <1121> <enter>
The number 1121 is not a palindrome. [Program Exits]
```

Q5. *Letter Conversion* [25 points]
Write code that asks the user to enter a character. Then the program asks the user to enter a non-negative integer offset. This program adds the offset to the character to produce a new ASCII value. This value should then be displayed. If the offset is 0 and the character is a letter, the program should change the case of the letter. *("Change case" means that a lowercase letter should be converted to uppercase, and an uppercase letter should be changed to lowercase.*) If the offset is 0 but the character is not a letter, the same character should be displayed.

Text in blue in the examples demonstrates the user inputs entered via the keyboard.

```
Sample runs:

Enter character: <D>
Offset (enter 0 to convert case): <7>
New character: K
```

```
Enter character: <a>
Offset (enter 0 to convert case): <0>
New character: A

Enter character: <B>
Offset (enter 0 to convert case): <0>
New character: b

Enter character: <3>
Offset (enter 0 to convert case): <0>
New character: 3
```

Hint: In the ASCII table (google it; easy to find), uppercase letters appear before lowercase letters. The offset between any uppercase letter and its corresponding lowercase letter is the same. You can compute this offset by:
int offset = ‘a’ – ‘A’;
or by: int offset = ‘A’ – ‘a’;

Notice that your program should error check if the resulting char code (original code + offset) is greater than X (you need to figure out X). You’ll notice in the ASCII table that this does not make sense. Think about why X is the limit.

Submission Details
Each program should be a single file. Please use the file names **Q1.cpp, Q2.cpp, etc.** for questions 1-5, respectively. Make sure to **comment** your code. **We will provide more information on how to submit this to GitHub** (where all PAs for this semester will be submitted). Do **NOT** submit your executable files (a.out or others). We only need the source code.