COSC 1437 - Fall 2016
Program Set #2

See 1437 Grading /Program Guide Sheet for directions and grading/submission information.

1. Implement Programming Exercise #8-Chapter 13 (Gaddis), pp.805. Use the member variables/functions as mentioned in the text. Place all code into one file. Output should be user friendly.

Name the program: TestCircleXX.cpp, where XX are your initials.

2. Every September, International Talk Like a Pirate Day holiday is celebrated. Write a simple C++ pirate to English translator program to celebrate this occasion. Here are some simple translation rules to follow:

- Drop yer 'g's and drop yer 'v's. You be talkin' pirate now, o'er here. Replace em with apostrophes.
- Hey -> Ahoy
- Of -> o'
- You -> ye, your -> yer,
- No more "My..." It be "Me..." from here on in.
- May all yer sentences end with ", matey", but keep the same endin' punctuation they woulda had, matey.
- Arr, me hearty, and may all yer sentences be'in with "Arr, me hearty,".
- There be no "are"s or "is"s in pirate talk. They all be "be"s.

Enter quit t' end, matey. Assume valid input from the keyboard. Output should look similar to below.

Sample Runs:

```
Enter yer input: Welcome aboard!
Arr, me hearty, welcome aboard, matey!

Enter yer input: I sense a mutiny on board.
Arr, me hearty, I sense a mutiny on board, matey.

Enter yer input: Hey my name is James Chegwidden of Texas.
Arr, me hearty, ahoy me name be James Chegwidden o' Texas, matey.

Enter yer input: quit
```

Name the program: PirateTalkXX.cpp, where XX are your initials.

3. Write a C++ program that reads an arithmetic expression written in English, such as

```
NINE PLUS SEVEN
```

and that outputs the number (integer) that is the result of evaluating the expression. More specifically, the input will be a one-digit number, written as a word in upper case letters, followed by a space, followed by one of the words "PLUS", "MINUS", "TIMES", or "DIVIDED-BY", followed by a space, followed by another one-digit number, written as a word in upper case letters. Here, "DIVIDED-BY" means integer division (Example: 8 divided by 3 is 2). The spellings for one-digit numbers are as follows: "ZERO", "ONE", "TWO", "THREE", "FOUR", "FIVE", "SIX", "SEVEN", "EIGHT", and "NINE". Assume that no input expression will call for division by zero, and valid input from the keyboard. The program should repeat until the user enters an expression that evaluates to zero. Output should look similar to below.

Sample Runs:

```
Enter the expression: NINE PLUS FOUR
Result: 13

Enter the expression: FIVE MINUS SEVEN
Result: -2

Enter the expression: ZERO TIMES SIX
Result: 0
```

Name the program: `MathWordExpressionsXX.cpp`, where XX are your initials.

4 (**). Write a C++ program to play the game of Three Musketeers--human against computer.

https://en.wikipedia.org/wiki/Three_Musketeers_%28game%29

The human should be able to choose which side to play: the Musketeers or Cardinal Richelieu's men. Once chosen, the human plays the same side for the entire game. The players take turns moving one piece; the musketeer player starts. The rules are as follows:

- The musketeer player can move a musketeer to any orthogonally (non-diagonal) adjacent space occupied by an enemy; the enemy piece is removed from the game.
- The enemy can move one enemy piece to any orthogonally adjacent empty space.

The enemy wins if it can force the three musketeers to be all on the same row or column. The musketeers win if on their turn they cannot move due to there being no enemy pieces adjacent to any musketeer and they are not all on the same row or column. As long as one musketeer can move, the game is not won.

After each move--computer or human--the program should print out the new game board. This could be just a display of the pieces, or it could be a display with rows and columns labeled like this:

```
    A B C D E
  -----------
1 | R R R R M
2 | R R R R R
3 | R R M R R
4 | R R R R R
5 | M R R R R
```

In addition, after each move, you should print a message telling what piece moved where. Don't make the user compare two game boards to figure out what just happened. Implement a way for the human to enter moves. When the program is run, it must explain how to enter moves; do this once, when the game is first started. The easier it is to use your program, the more fun it will be to play. Of course, the computer should prevent the human from making illegal moves. It should also tell when the game is over, and who won.

Implementation:

This program will be less structured than previous assignments. Some classes that will be needed are specified and additional classes can be used. It is up to the student to figure out what the responsibilities (methods) of each class will be. Those classes are:

- `Musketeer` (object) - to represent the entire game. Creating a `Musketeer` object should start the game.
- `Player` (object) - to represent a single player.

Also, no object should directly examine or alter the variables of any other object; all access should be done by talking to (calling methods of) the other object. The student should figure out what the various classes and objects should be and what their instance variables and their methods should be. There is no single "correct" design; any reasonable design will do. The better the design, the easier the programming will be. Use the objects defined to play a game of Three Musketeers. The computer player must play legally, but does not need to play well. Try to make the computer player "smart," so the human player does not always win. Place all code into one file. Output should be user friendly.

Name the program: `OOMusketeersXX.cpp`, where `XX` are your initials.