Last updated 05 November, 2014

Assignment 7 (100 points):

Please see Submission Checklist (below) for submission requirements.

It is time to bring together some concepts we have been learning!

Remember that pointers can be tough to think about, so draw some out with all the parts of a pointer on paper to get an idea firmly in your mind about how they work!

(10 points) Programming Styles and Convention.
http://classes.engr.oregonstate.edu/~jessjo/CS161/OSU_IntroCodeStandards_v1.1.pdf

(30) Remember to submit your report (just over the project, not the exercises)!
Which includes:

- Understanding: a description of what you understand is being asked of you for the assignment.
- Design: an outline in **pseudocode or a drawing** as to how your code will be implemented. The design section is to be done before you start coding.
- Testing: a description of the tests you will implement to ensure your code works along with the actual testing. The testing section is to be done before you start coding. **(Make a table with three columns named Input, Expected Output and Actual Output. Discuss any discrepancies.)**
- Reflection: a discussion of the problems you encountered and how you solved them.

(20) Exercise components:

1. (10) Write a function that takes as parameters a square 2D array of ints and the number of rows in the user's square. The function should return a boolean value indicating whether all of the rows, columns, and both major diagonals of the user's square sum to the same value. In main, you should ask the user for the size of their square. You can assume a maximum size of 10x10 – the user's square doesn't have to fill the array. Then let the user enter values to fill their square. Let them enter a whole row on one line, with whitespace separating the values. After that, call your function and then display the return value. The result for each of the examples below should be *true*:

```
1 1 1 1     11 24 7  20 3      1 3 2
1 1 1 1     4  12 25 8  16     3 2 1
1 1 1 1     17 5  13 21 9      2 1 3
1 1 1 1     10 18 1  14 22
            23 6  19 2  15
```

File must be called: sameSum.cpp

[I'm having you assume a maximum size because dynamically allocating a 2D array is somewhat involved. You can do it, but either the number of columns has to be known at compile time, or you have to use a loop involving a double pointer to make an array of arrays (which is really what a two-dimensional array is) and then when you're done, use a loop to delete them all. Another option would be to use a one-dimensional array with the number of elements you want and use math to pretend that it's a two-dimensional array.]

2. (10) Write a program that allows the user to use command line arguments to specify the sizes of 3 int arrays they will enter data for. Then you will **dynamically allocate the arrays using *new***. Then allow the user to enter the integers for each element in each array. Let them do this on a single line with whitespace separated values. Next, for each array, print out whether it is symmetric. That is, the first element is equal to the last element, the second element is equal to the next-to-last element, and so on for the rest of the array (you should make a function to handle this, which takes an array and its size and then returns a boolean value). Lastly, you will **deallocate the arrays with *delete***. For example:

```
symArrays 5 3 4

Please enter the numbers for the first array.
34 19 12 19 34
```

```
Please enter the numbers for the second array.
1 2 3
Please enter the numbers for the third array.
2 3 3 2

Array 1 is symmetric.
Array 2 is not symmetric.
Array 3 is symmetric.
```

Error checking: If there are not exactly three command line arguments (four counting the name of the program), print an error message and end. If the number of data a user enters on a line doesn't match their specified size for that array, print an error message and let them try again. You do not have to validate that arguments or data are of the correct type, though you can if you wish.

File must be called: symArrays.cpp

(40) Project component

Write a program that allows two people to play a game of tic-tac-toe. Use a 3x3 array to represent the board. Display which player's turn it is, allow them to specify a square (as a pair of coordinates, first row then column), and display the new state of the board. If one of the players makes a winning move, declare that player the winner. If the board becomes full and neither player has won, declare it a tie. When running your program, allow the user to specify how many games they want to play as a command line argument. Your program should let them play that many games and then display how many games each player has won and declare the overall winner (or tie). If the user does not specify a command line argument the program should just play one game. In a series of games, players X and O should switch who plays first.

```
  0 1 2
0 . . .
1 . . .
2 . . .

Player X: please enter your move.
0 0

  0 1 2
0 x . .
1 . . .
2 . . .

Player O: please enter your move.
1 2

  0 1 2
0 x . .
1 . . o
2 . . .

Player X: please enter your move.
1 2
That square is already taken.
  0 1 2
0 x . .
1 . . o
2 . . .

Player X: please enter your move.
```

Error checking: If someone tries to take an occupied square, tell them that square is already occupied and ask for a

different move.

File must be called: ticTacToe.cpp

Remember to submit your **report** and **source files** to TEACH before the end of Sunday.

Remember to keep discussion going!

Submission Checklist:

makefile

Exercises (20 pts):
- sameSum.cpp
- symArrays.cpp

Project (40 pts):
- ticTacToe.cpp

Report, ***in PDF format*** (30 pts):
- Understanding
- Design
- Testing
- Reflection

The implementation part of the project is the .cpp file you submit.