CS 1410 Introduction to Computer Science – CS2
Assignment #2: Pointer Programming

Given: Saturday, January 17, 2015
Due: 1:00 p.m. Monday, January 26, 2015
Total Points: 50 points

Problem Description

The purpose of this assignment is to give more practice in working with pointers and dynamic memory. You are required to write a Connect Four like game (Connect N game), except that your game will allow the user to select anywhere from 3 to 5 for the size of "Connection" for displaying and debugging purpose. You can find an example of the Connect Four game at the following web site: http://www.mathsisfun.com/games/connect4.html

The program starts by asking the user how many connected pieces are needed to win. This number should be 3, 4, or 5. The game board has 3 more columns than the number of connected pieces to win and 2 more rows. For example, if the user selects 5 as the number of connected pieces to win, the game board has 7 rows by 8 columns. Note: The grader will use any random connected pieces other than 3, 4, and 5 to test your program.

Here are some requirements about the program:

- The first player is the user and the second player is the computer. The computer randomly chooses which column to drop a piece on.
- Use the letter 'p' for the player and the letter 'c' for the computer.
- Draw the game board between each turn.
- Between each turn, your program must check to see if there is a winner or a tie.
- Once there is a winner, indicate if it is the player or the computer and quit the program; before quitting, be sure to release any memory that has been allocated for the game board.

Programming Tasks

Design and implement the Connect N game with the following instructions:

1. Declare a structure Position which contains the position (e.g., row and column number) of the current piece, which can be from the player or the computer. (Refer to Chapter 7.12 for more examples on the use of structures).

2. Declare a class ConnectNGame to store the number of connected pieces required to win and the game board. **You may add more data members if you wish.** To create the game board, you have to dynamically allocate a two-dimensional char array. Because you don't know how many rows or columns there might be, you have to use a "double pointer", a char **. This is a "pointer to a pointer." The first memory allocation is to allocate an array of char pointers that is the number of rows in the game board, the second set of memory allocations is to allocate the char array for each of the rows, that is the number of columns in the game board. Once allocated, you can use the array just like a normal two dimensional array. **However, in this assignment, you have to use * (indirection operator) to access (read and write) each element of the two dimensional array.** (Refer to Chapter 10.3 for more examples on the use of indirection operator)
   For example, you may declare the game board as follows:
   char **gameBoard ;
   You can then perform the following two steps to dynamically allocate memory spaces.

```
Step 1) gameBoard = new char *[rows];
            // rows is a variable storing the number of rows of 2-D array
Step 2) use loop to allocate the char array for each of the rows.
for (int count = 0 ; count < rows ; count ++)
     gameBoard[count] = new char [cols] ;
      // cols is a variable storing the number of columns of 2-D array
```

3. **[10 points] This ConnectNGame class must have the following three public methods.**
   /* Constructor: Create a new board given the number of connected pieces required to win. Initially, the new board contains empty spaces (e.g., ‘ ’).*/
   [2 point] ConnectNGame(int);

   /* Destructor: Clean up the game board. */
   [2 point] ~ConnectNGame() ;

   /* Play the game and quit when there is a winner or tie */
   [6 points] void Play() ;

4. **[35 points] This ConnectNGame class must have the following private methods.** However, you may add other private methods as you feel appropriate.

   /* Provide a pretty display of the game board */
   [2 points] void DisplayGameBoard() ;

   /* Ask the player for their move. Specifically, it asks the player for the column that the player wishes to drop his piece into and sets the value of the lowest row of that column that is not occupied by a 'p' or a 'c' to 'p'. It will return the position of the player’s piece. */
   [4 points] Position PlayerMove() ;

   /* Randomly generate the computer move. Specifically, it uses rand() function to generate the column that the computer wishes to drop his piece into and sets the value of the lowest row of that column that is not occupied by a 'p' or a 'c' to 'c'. */
   [4 points] Position ComputerMove() ;

   Hint: You may use the following to initialize the random number generator. Each time different random numbers will be generated by **rand()** after this initialization.
   srand(static_cast<unsigned int>(time(NULL)));
   rand() ;

   /* Check for a victory by horizontally-laid pieces. If there are N game pieces in a continuous horizontal line, the last player that placed pieces wins. */
   [5 points] bool CheckRowWin(Position) ;

   /* Check for a victory by vertically-laid pieces. If there are N game pieces in a continuous vertical line, the last player that placed pieces wins. */
   [5 points] bool CheckColumnWin(Position) ;

   /* Check for a victory by diagonally-laid pieces, rising to the right. If there are N game pieces in a continuous diagonal line, the last player that placed pieces wins. */

[5 points] bool CheckRightDiagonalWin(Position)

/* Check for a victory by diagonally-laid pieces, rising to the left. If there are N game pieces in a continuous diagonal line, the last player that placed pieces wins. */
[5 points] bool CheckLeftDiagonalWin(Position)

/* Check for a tie. If all columns are filled and there are not any N game pieces in a continuous horizontal, vertical, and diagnoal line, no player is the winner. */
[5 points] bool CheckTie(Position) ;

5. You should write your program and break out the definitions and implementations into main.cpp, ConnectNGame.cpp, ConnectNGame.h.

   [2 points] main.cpp: Implement the main function to prompt the user for how many connections to win the game. The number of connections must be 3, 4, or 5. If the user's input is wrong, your program will keep asking the users for the input until the correct information is provided. This main function then creates an instance of the ConnectNGame and call Play function to animate the game.

   [3 points] ConnectNGame.h: Contain the private data members and function prototype for each function member.

   [45 points] ConnectNGame.cpp: Implement all functions defined in ConnectNGame.h.