CS 1410 Introduction to Computer Science – CS2
Section 1: MWF 10:30 a.m. – 11:20 a.m.
Section 2: MWF 1:00 p.m. – 1:50 p.m.
Instructor: Xiaojun Qi
Assignment #2
Given: Thursday, January 16, 2014
Due: 11:59 p.m. Monday, January 27, 2014
Total Points: 30 points

The purpose of this assignment is to give more practice in working with pointers and dynamic memory. You are required to use C++ to write a Connect Four like game (Connect N game), except that your game will allow the user to select anywhere from 3 to 6 for the size of "Connection". 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 in the range of 3 to 6. That is, the number should be 3, 4, 5, or 6. 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. **To create the game board, you need (and are required) 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.

Some notes 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 'o' for the player and the letter 'x' for the computer.
- Draw the game board between each turn.
- Between each turn your program must check to see if there is a winner
- Once there is a winner, indicate if it is the player or the computer and quit the program; before quitting, be sure to deallocate any memory that has been allocated.
- Use functions for each major component of the program: e.g., a function to allocate and initialize the game board, a function to test for a winner, a function to display the game board, a function to deallocate the game board, etc.

```
/* 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() ;
```

Here are some sample function prototypes for your reference. You may design your own function prototypes at your will.

```
/* Prompt the user for how many connections to win the game. The number of connections must
be 3, 4, 5, or 6. */
int GetConnectionSize(void) ;
```

```
/* Create a dynamic 2D character array for the game board */
char** BuildGameBoard(int WinSize) ;
```

Hint:
Step 1) char **GameBoard = new char *[WinSize+EXTRA_ROWS];
Step 2) use loop to allocate the char array for each of the rows.

```
/* Provide a pretty display of the game board */
void DisplayGameBoard(char **GameBoard, int WinSize) ;
```

```
/* 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 an 'x' or an 'o' to 'o'. */
void PlayerMove(char **GameBoard, int WinSize) ;
```

```
/* 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 an 'x' or an 'o' to 'x'. */
void ComputerMove(char **GameBoard, int WinSize) ;
```

```
/* 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. */
bool CheckRowWin(char **GameBoard, int WinSize) ;
```

```
/* 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. */
bool CheckColumnWin(char **GameBoard, int WinSize) ;
```

```
/* 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. */
bool CheckRightDiagonalWin(char **GameBoard, int WinSize)
```

```
/* 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. */
bool CheckLeftDiagonalWin(char **GameBoard, int WinSize)
```

```
/* 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. */
bool CheckTie(char **GameBoard, int WinSize) ;
```

```
/* Clean up the game board. That is, deallocate the game board from memory */
void DestroyGameBoard(char **GameBoard, int WinSize) ;
```