Assignment 5 Connect Four

Design Due: Sunday, 12/1/19, 11:59pm on Canvas

Peer Review 5 Due: Friday, 12/6/19, 11:59pm on Canvas

Code Due: Sunday, 12/8/19, 11:59pm on TEACH

Make sure you demo Assignment 4 within two weeks of the due date to receive full credit. If you go outside the two-week limit without permission, you will lose 50 points. If you fail to show up for your demo without informing anyone, then you will automatically lose 10 points.

Introduction
This particular program implements a game that is played in real life. For this assignment you should try to make your program look as nice as possible using the provided code fragment.

This assignment will not be demoed. TAs will grade on their own during final's week!!!

Problem statement:
Write a C++ program that plays the game of **Connect Four**. The game is simple and you can view the instructions online here.

You can also find online implementations if you want to see the game in action. One specific online implementation is at: https://www.mathsisfun.com/games/connect4.html.

Your game will allow 1-2 players, and at the end, you need to ask if the user(s) want to play again. In this program, we are going to use "X"s and "O"s to represent pieces for different players. Those pieces are inserted into a board and the first player to get 4 adjacent pieces (horizontally, vertically, or diagonally) wins.

Game Set up
Command line arguments will be used to indicate the number of players as well as the size of the board. The three command line arguments will be provided in the following order: **`number of players, number of columns, number of rows`**. **Neither the number of columns nor the number of rows can be over 20!!!** A two-player game implies that two humans will be present and the program will allow each player to take turns. A one-player game implies that the human is playing against the computer (see One-Player Operation for more details).

Example command to start a two-player game with 7 columns and 6 rows:
```
./connect_four 2 7 6
```

If you want to start a one player game on a 9 x 7 grid, you would use the following command:
```
./connect_four 1 9 7
```

- **Two-Player Operation**
  In two-player mode, your code will first display the empty board (with each column numbered across the top). It will then prompt the first player to select a column. After a column is selected, the screen will display the updated board with the player's piece at the bottom of the selected column. Player two can then choose a column in which to drop their piece. This behavior continues (alternating between players) until a winner is determined or until no more pieces can be dropped into the board (resulting in a tie).

- **One-Player Operation**
  In one-player mode, the human will play against the computer. Your code will first ask the human if they want to have the first move. If so, the human gets to drop the first piece. If not, the computer gets to drop the first piece. In order to make this programming assignment easier, the computer player does not have to be intelligent. For each computer move, the program will randomly select a column and drop the piece. Naturally, the computer player can only drop a piece into columns that have at least one empty space remaining. As with the two-player operation, game play will alternate between each player.

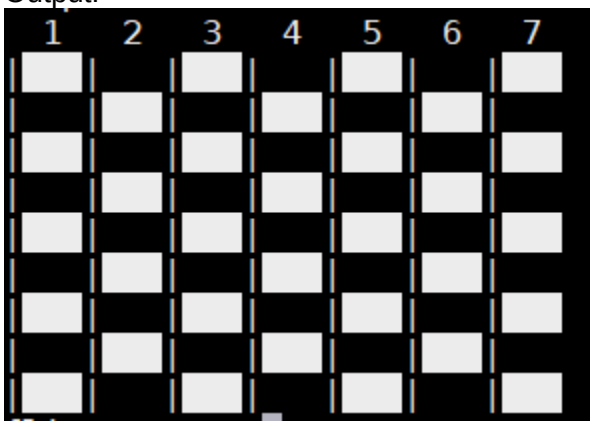
Implementation Requirements

In addition to the earlier specifications, your program must meet these requirements:

- You must use a dynamic 2-dimensional array to represent the board
- Establish the size of the board via command line arguments, must include error handling for too many and too few arguments as well as incorrect input (negative number, floating point number, text string, etc). If an invalid value is provided, then the program should display a message to indicate the problem, and recover by asking for these values during runtime.
- The board must be correctly colored black and white using the following code as a base. It also needs to display the column numbers across the top. The following code fragment colors a two-dimensional board. It is expected that you will adjust it as needed to provide the best user interface possible for your program.

```
for (int i=0; i<rows; i++)
{
      for (int j=0; j<cols; j++)
      {
            if (i % 2 == 0 && j % 2 == 0)
                  cout << "|\033[30;47m " << board[i][j] << " ";
            else if (i % 2 == 1 && j % 2 == 1)
                  cout << "|\033[30;47m " << board[i][j] << " ";
            else
                  cout << "|\033[0m " << board[i][j] << " ";
            cout << "\033[0m";
      }
      cout << endl;
}
```

Output:



- Your program must display the updated board after each move.
- If a winner exists, the program must immediately declare the winner and ask if the player(s) want to play again.
- If no more moves are possible and no winner exists (i.e. the entire board is full of pieces), the

program must declare the game a tie and then prompt the player(s) to start a new game.
- Print an error message and recover when the player supplies an invalid column. This could be a column that doesn't exist ("Cat", -4, 142, etc) or it could be a column that is already full of pieces.
- Play the game correctly based on rules and number of players.
- The computer player must follow the rules of the game and can only drop pieces in columns that have at least one open space.

Submission Information
Assignment 5 will not be demoed. TAs will grade on their own during Finals Week. Besides your code, you will submit a README.txt that outlines how your program is compiled and run. Failure to submit a README.txt will result in a deduction as well as any penalties that may be incurred as a result of incorrect use of your program.

Extra Credit (10 pts) – Implementing a smarter Computer Opponent
Instead of using a simple random number generator to select the column, write your computer player implementation so that it has more intelligence. **If you choose to attempt this extra credit be sure to document your algorithm in the README.txt as well as to display this information on the screen to the user).**
As you can imagine, there are many ways to implement a smarter computer opponent. The options are endless!

Design Document – Due Sunday 12/1/19, 11:59pm on Canvas
Refer to the Example Design Document – Example_Design_Doc.pdf

Understanding the Problem/Problem Analysis:
- What are the user inputs, program outputs, etc.?
- What assumptions are you making?
- What are all the tasks and subtasks in this problem?

Program Design:
- What does the overall big picture of each function look like? (Flowchart or pseudocode)
    - What data do you need to create, when you read input from the user?
    - How to name your variables?
    - What are the decisions that need to be made in this program?
    - What tasks are repeated?
    - How would you modularize the program, how many functions are you going to create, and what are they?
- What kind of bad input are you going to handle?

Based on your answers above, list the **specific steps or provide a flowchart** of what is needed to create. Be very explicit!!!

Program Testing:
Create a test plan with the test cases (bad, good, and edge cases). What do you hope to be the expected results?
- What are the good, bad, and edge cases for ALL input in the program? Make sure to provide enough of each and for all different inputs you get from the user.

Electronically submit your Design Doc (.pdf file!!!) by the design due date, on Canvas.

Program Code – Due Sunday, 12/8/19, 11:59pm on TEACH

Additional Implementation Requirements:

- Your user interface must provide clear instructions for the user and information about the data being presented
- Use of 2D dynamic array is required.
- Use of command line arguments is required.
- Your program must catch all types of error and recover from them.
- You are not allowed to use libraries that are not introduced in class.
- Your program should be properly decomposed into tasks and subtasks using functions. To help you with this, use the following:
  - Make each function do one thing and one thing only.
  - No more than 15 lines inside the curly braces of any function, including main(). Whitespace, variable declarations, single curly braces, vertical spacing, comments, and function headers do not count.
  - Functions over 15 lines need justification in comments.
  - Do not put multiple statements into one line.
- You are encouraged to use the functions in assignment 2 to do error handling.
- No global variables allowed (those declared outside of many or any other function, global constants are allowed).
- You must not have any memory leaks
- You program should not have any runtime error, e.g. segmentation fault
- Make sure you follow the style guidelines, have a program header and function headers with appropriate comments, and be consistent.

Electronically submit your C++ program (.cpp file, not your executable!!!) by the code due date, on TEACH.