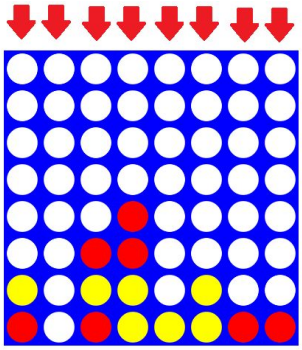

Connect Four

Project Profile

The goal of this project is to implement the game Connect Four using the C programming language. The game will be interactive and played between a human user and an AI (artificial intelligence). The focus of this project is primarily on exercising an introductory understanding of the C programming language including basic data types, looping and conditional constructs, arrays, iteration, basic I/O, formatted output, and functions. This semester, all projects are individual work (not done in groups), unlike the challenges and labs!

Connect Four

Traditionally, the connect four game is a two-player connection game in which the players first choose a color and then take turns dropping one colored disc from the top into a seven-column, six-row vertically suspended grid. The pieces fall straight down, occupying the lowest available space within the column. The objective of the game is to be the first to form a horizontal, vertical, or diagonal line of four of one's own discs. Please see the Wikipedia entry for more details. In your implementation you will use X and O characters in place of the colors.

Game Board

The game board is a 7-column by 6-row suspended grid. The rows are labeled 1 - 6 and the columns are labeled A - G. This is an example of an empty game board:

```
  A B C D E F G
6 . . . . . . .
5 . . . . . . .
4 . . . . . . .
3 . . . . . . .
2 . . . . . . .
1 . . . . . . .
```

The rows are labeled from the bottom to top starting with 1, and the columns are labeled from left to right starting with A. Here is an example state of the game where play begins with the human player using the X discs (X always plays first):

```
  A B C D E F G
6 . . . . . . .
5 . . . . . . .
4 . . O . . . .
3 . . O X . . .
2 . O X O X . .
```

```
1  . O X X O X .
```

The record of moves can easily be recorded as `[D,E,C,B,F,B,C,C,E,D,D,C]` where the first entry is an X move and then alternates between O and X on each subsequent entry.

Game Play

Your implementation will prompt the user before game play to determine who goes first and then proceed to alternate between the human player and the AI. When it is the human's turn the game will prompt the player for which column she would like to drop in a disc. For example, using the above game state, if the human player chose column B the resulting game state would be:

```
   A B C D E F G
6  . . . . . . .
5  . . . . . . .
4  . . O . . . .
3  . X O X . . .
2  . O X O X . .
1  . O X X O X .
```

The next move would be decided by the AI player, followed by prompting the human player again. Game play would then proceed in that fashion. Here is an example of the game playing out to completion:

A B C D E F G 6 . . . . 5 . . . . 4 . . O . . . . 3 . X O X . . . 2 . O X O X . . 1 . O X X O X .	A B C D E F G 6 . . . . 5 . . . . 4 . . O . . . . 3 . X O X . . . 2 . O X O X . . 1 **O** O X X O X .	A B C D E F G 6 . . . . 5 . . . . 4 . . O . . . . 3 . X O X . . . 2 . O X O X . . 1 O O X X O X **X**	A B C D E F G 6 . . . . 5 . . . . 4 . . O O . . . 3 . X O X . . . 2 . O X O X . . 1 O O X X O X X

In this example, the AI player got *connect four* in a diagonal with discs 1A, 2B, 3C, 4D and won the game.

Requirements

Your primary objective is to use the C programming language to design and implement the connect four game that operates according to the game play described in the previous section. You are required to design and implement the appropriate data structures and corresponding algorithms that will enable a human player to play against an AI player. The AI design is entirely up to you, however, it should be at least as smart as a 5 year old playing the game. That is, the AI should make some obvious choices such as blocking as a defensive move or trying to build a connection (horizontal, vertical, or diagonal) as an offensive move. If you have taken a course in Artificial Intelligence you are welcome to make your AI more sophisticated (e.g., Minimax), however, this is not required. Furthermore, you are constrained to using only the C constructs that we have covered up through the first week of class. In particular:

Allowable C Language Constructs	
• printf	• 1D array and initialization • const

• basic C data types and variables (int, float, double, char, _Byte/byte) • storage sizes and ranges • type specifiers • arithmetic expressions • for, while, do loop, if statement, switch, conditional operator • aligning output • scanf	• multi-dim arrays • variable length arrays • array length bounds, iteration, length • functions, arguments, locals, returns • prototype declaration • functions and arrays, mutability • global variables • automatic/static variables

In addition, your implementation must meet the following specific requirements:

1. An array must be used to represent the game board as described above.
2. One or more functions must be used as part of the implementation.
3. Arrays must be passed to functions as arguments.
4. Iteration must be used to traverse the game board.
5. Your implementation must be able to determine end of game: win, lose, or draw.
6. An AI must exist in your game as described above.
7. You should minimize the use of global variables - no global variables is the best.
8. Your implementation should allow either the human or AI to go first.
9. Your implementation must show the state of the game, modeled after our examples above, after each move (both human and AI) and display the final game state, who won or who drew, along with the recorded moves in the form mentioned in the previous section. All output should be properly formatted and aligned.

Video Demonstration

You must provide a link to a 1-minute video demonstration of your game being played. Your video should be short and get to the point, showing a game being played to completion. That is, displaying each alternating game state, final state, and who won (or if it was a draw). You must provide a voice in the video demonstration (silence is not allowed); explain what you are doing when you play the game.

Grading and Rubric

You will be graded according to the following rubric. The scoring of the rubric below falls under various categories and headings. Categories may also be multiplied - that is, some categories count more than others (e.g, x2, x3).

	Exceeding = 4	Meeting = 3	Approaching = 2	Beginning = 1	No Submission = 0
Delivery x 3	• Completed between 90-100% of the requirements • Delivered on time, and in correct format	• Completed between 80-90% of the requirements • Delivered on time, and in correct format	• Completed between 70-80% of the requirements • Delivered on time, and in correct format	• Completed less than 70% of the requirements. • Delivered on time, and in correct format	• No submission
Coding Standards	• Excellent use of white space • Creatively organized work • Excellent use of variables (no or minimized use of	• Good use of white space • Organized work • Good use of variables (no or minimized use of global variables,	• White space makes program fairly easy to read • Organized work • Good use of variables (few global variables,	• Poor use of white space (indentation, blank lines) • Disorganized and messy • Poor use of	• No submission

	global variables, unambiguous naming)	unambiguous naming)	unambiguous naming)	variables (many global variables, ambiguous naming)	
Modularity	● Excellent demonstration of modular design (files and/or functions are carefully organized into sensible units)	● Good demonstration of modular design (files and/or functions are organized into mostly sensible units)	● Basic demonstration of modular design (files and /or functions are partially organized, organization could be improved)	● Poor demonstration of modular design (files and/or functions are minimally organized, unclear organization)	No submission
Data Structures	● Excellent organization of data structures and their relationships ● Excellent organization of data amongst data structures	● Good organization of data structures and their relationships ● Good organization of data amongst data structures	● Basic organization of data structures and their relationships ● Basic organization of data amongst data structures	● Poor organization of data structures and their relationships ● Poor organization of data amongst data structures	● No submission
Code Documentation	● Clearly and effectively documented including descriptions of all variables ● Specific purpose is noted for each function, control structure, input requirements, and output results	● Clearly documented including descriptions of all variables ● Specific purpose is noted for each function and control structure	● Basic documentation has been completed including descriptions of all variables ● Purpose is noted for each function	● Minimal documentation has been completed ● Purpose of some functions noted	● No submission
Runtime x 2	● Executes without errors, excellent user prompts, good use of symbols, spacing in output	● Executes without errors ● User prompts are understandable, minimum use of symbols or spacing in output	● Executes without errors ● User prompts contain little information, poor design	● Does not execute due to errors ● User prompts are misleading or non-existent	● No submission
Efficiency	● Solution is efficient, easy to understand, and maintain	● Solution is efficient and easy to follow (i.e. no confusing tricks).	● A logical solution that is easy to follow but it is not the most efficient	● A difficult and inefficient solution	● No submission
General Knowledge	● Solution demonstrates mastery of knowledge with respect to the domain of the problem ● No spurious or	● Solution demonstrates solid knowledge with respect to the domain of the problem ● No spurious or unused code	● Solution demonstrates basic knowledge with respect to the domain of the problem ● Some spurious or unused code	● Solution demonstrates lack of knowledge with respect to the domain of the problem ● Spurious or	● No submission

	unused code exists, every aspect of the code provides a purpose for the overall solution without extraneous work	exists, every aspect of the code provides a purpose for the overall solution without extraneous work	exists, not every aspect of the code provides a purpose for the overall solution	unused code exists in several parts of the solution, not every aspect of the code provides a purpose for the overall solution, difficult to follow	
Demonstration	● Demonstration of solution is excellent, clearly shows the application running according to the requirements	● Demonstration of solution is good, clearly shows the application running according to most of the requirements	● Demonstration of solution is basic, does not clearly show the application running according to most of the requirements	● Demonstration of solution is poor, does not clearly show the application running according the requirements	● No submission

Submission

You must submit two files to Gradescope by the assigned due date:

- **`cf.c`** - this is your implementation of the connect four game.
- **`README.txt`** - this is a text file containing a brief 1 paragraph overview of your submission highlighting the important parts of your implementation. The goal should make it easy and obvious for the person grading your submission to find the important rubric items. This text file should also clearly include a URL to your video for us to review.

Make sure you submit this project to the correct assignment on Gradescope! Remember, projects are to be done individually, not in groups!.

You are required to make your program runnable in the VM environment we give you.