# Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2016

## Homework 6 – Connect-3 Game with Object Sharing

Due: 20/04/2016, Wednesday 21:00

**PLEASE NOTE:**

**Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!**

**You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!**

### Introduction

In this homework, you are asked to implement a simple board game using *object sharing* concept of C++. Main function of the implementation, which includes the game implementation using classes, is given to you and you are expected to write two classes: *Board* class and *Player* class. We are going to explain these classes in more details in the following sections. Before that, we start with a general description of the game.

### The Game

The game is a simplified version of the well-known two-player connection game Connect-4. In our version, which is called Connect-3, the game board is a $5 \times 4$ character matrix (of **char** type). The game is played by two players and each player has a game piece, which is represented as a character in the program. The board is a *vertical* one and players drop their pieces through the columns. A piece *falls* straight down in the column and will be placed in next available cell. Basic gravity rules apply here; i.e., if the column is empty, the first piece will be in the bottom of the column; however, if there are some pieces in the column, the next one will occupy the cell above the last piece of the column. The players play in turns. In a turn, a player chooses a column to drop its game piece.

A player wins by connecting **three** of its own pieces straightly (vertically, horizontally or diagonally). The game may also end in a draw, if the game board is filled with pieces and none of the players has straightly connected three pieces.

Here is a sample iteration of the game which may help to illustrate the gameplay. The game begins with an empty board as shown below. Let us assume that p1 and p2 are playing, and p1 uses ‘X’ character as her game piece and p2 uses ‘O’ character.

| | | | |
|---|---|---|---|
| - | - | - | - |
| - | - | - | - |
| - | - | - | - |
| - | - | - | - |
| - | - | - | - |

Fig 1. Initial State of the Board

Let us also assume that p1 starts first. p1 can choose one of the four columns by choosing an integer between the interval [0, 3]. If p1 chooses column 2, the next state of the game board will be as follows:

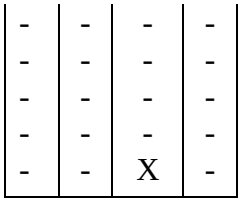


Fig 2. After p1 plays by selecting column 2

Then it will be p2's turn. If p2 chooses column 1, the board becomes as follows:

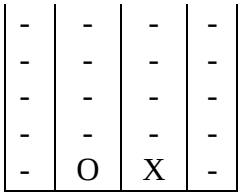


Fig 3.After p2's turn (selecting column 1 to play)

In the next turn, if p1 chooses column1as well, the game piece X will fall above the existing game piece in the column.

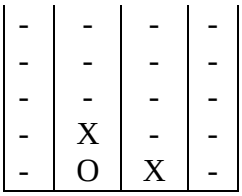


Fig 4.After p1 chooses column 1

Suppose in the next turn, p2 chooses column 1 again. This time, the game piece O will fall above existing pieces in the column.

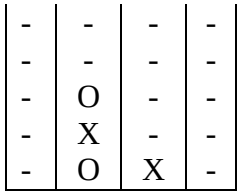


Fig 5. p2 chooses the same column again

Assume that players kept playing and let us fast forward to the point that p1 has won the game by connecting three of her own game pieces (X). At its final state, the board might look like below. Three connected game pieces X are highlighted.

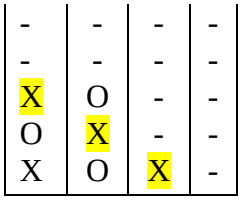


Fig 6. The final state of the board

## The `Board` Class

The `Board` class will be used to create and manipulate a board on which the game will be played. A board is represented by a built-in matrix, which is a private data member of the class. The matrix will have fixed number of rows and columns. You can define this private variable as shown below.

```
char myBoard[5][4];
```

Now, we will give constructor and some member function definitions of the `Board` class.

**Constructor:** You need to initialize all of the matrix elements to dash, '-', character which will indicate that the cell is empty.

**displayBoard:** The displayBoard function does not take any parameters. It only displays the current state of the board. You need to display the game board exactly as you will see when you run the executable file that comes with this document. Please remark that there should be an extra blank between two horizontal cells for a tidy output.

**dropPieceOn:** This function basically drops a piece on a column. It takes one `int` parameter and one `char` parameter, and returns a Boolean value. The integer parameter specifies the column number on which the piece will be dropped. The character parameter specifies the piece to be dropped. The function is supposed to return true if the piece is successfully dropped on the column of the board. Before doing anything, you have to check if this number is in column range. If not in rage, obviously you have to return *false*. Then you have to check if the given column has any free cell. If the column is completely filled with pieces, again you have to return *false*. Otherwise, you have to put the piece in its place as described in **The Game** section and return *true*.

**isFull:** This function does not take any parameters and returns a Boolean value. It will check if there are any free cells left on the board. If there is no space, it should return *true*; otherwise, it should return *false.*

**isConnect3:** This function takes a character parameter and returns a Boolean value. It will check if there is an occurrence of three straightly connected pieces of the parameter character (vertically, horizontally or diagonally). If there is such an occurrence, then the function should return *true*; otherwise, the function should return *false.*

Some of the above functions are explicitly used in game implementation given to you in main.cpp. However, some of them are not directly used in main.cpp; but these need to be used by the `Player` class member functions. Since the use of friend functions and friend classes are **not** allowed in this homework, in the implementation of the player class, you will need to use them to manipulate the shared board object.

## The `Player` Class

`Player` class will be used to manage the players of the game. There will be two players playing on the same board in a game. Thus, player objects must share a board object using *object sharing* concept and principles of C++ as we have seen in the lectures. We have seen two different methods for object sharing in the course; due to our main function implementation, which is provided to you, you must use the reference variable method.

The player class should keep its game piece in a `char` type of private data member. Now, we will give member function definitions of the `Player` class.

**Constructor:** Constructor of the player class takes two parameters, which are the board object that will be played on and the game piece character of the player. These parameters are used to initialize the corresponding private data members.

**play:** The play function takes a column number parameter and returns a Boolean value. The player will try to play its piece on the parameter column. The function should return *true* if the piece is placed successfully in the column; otherwise, it should return *false*.

**isWinner:** This function does not take any parameters. It will return *true* if the player has won the game; return *false* otherwise.

### Using Object Sharing Principles and Object Oriented Design

In your program, the *Board* object must be shared by the *Player* objects. For this object sharing, you have to employ the method that uses reference variables.

**It should be clear that you will write two classes for `Board` and `Player`. You need to analyze the requirements carefully and make a good object-oriented design for these classes. In this context, you have to determine the data members and design/implement member functions of each class correctly. We will evaluate your object oriented design as well. Moreover, you are not allowed to use friend class or friend functions in your design (i.e. you are not allowed to use the `friend` keyword anywhere in your code). Our aim by this restriction is not to make your life miserable, but to enforce you to proper object oriented design and implementation.**

### Provided files

**connect3.exe:** This is the executable file of our implementation for the board game. In this homework, we do not provide any sample runs due to the interactivity of the game. Instead we provide this executable so that you can try and understand the requirements.

**main.cpp:** This file contains the `main` function and the related code, which contain the game implementation by using related class functions. In this homework, our aim is to reinforce object oriented design capabilities; thus, we did not want you to deal with the class usage, but focus on their design and implementation. Please examine this provided file in order to understand how the classes are used and how the game is played. We will test your codes with this main.cpp with different inputs. You are not allowed to make any modifications in this file (of course, you can edit the file to add `#includes`, etc. to the beginning of the file); you have to create and use other files for class definitions and implementations.

## Please see the previous homework specifications for the other important rules and the submission guidelines

Good Luck!
Albert Levi, Ömer Mert Candan