CORK INSTITUTE OF TECHNOLOGY
INSTITIÚID TEICNEOLAÍOCHTA CHORCAÍ

Computing@CIT

C PROGRAMMING

Assignment 2: Connect-N Game Manager.
Part 1: Heap Memory and Strings.

1. Background:

Websites as MINICLIP or Agame (http://www.miniclip.com/games/en/ and (http://www.agame.com/, respectively) support online communities of users playing to single and multiplayer-based small games. Among their functionality they include a social component classifying and ranking both games and users.

It is out of the scope of this module to understand how these websites operate internally. However, given our C knowledge, we are going to create a small C application replicating some of their functionality. More specifically, our off-line, local and console-based application is going to support a community of users playing to the classical game Connect4 (https://en.wikipedia.org/wiki/Connect_Four) and variants of it.

To this end, we are going to split the assignment into two parts:

- Part1: Management of a single game.
- Part2: Management of a community of users, open games and data storage.

2. Game Logic:

In this first part of the assignment we are going to manage a single game, which we fully encode under the user-defined datatype `struct _game` (type-aliased to the handier name `game`).

The different fields or pieces of information that we want to represent are:

1. **long id** → Unique identifier for each game.
2. **char* p1** → Name of first player.
3. **char* p2** → Name of second player.
4. **int mode** → Game mode being played. There are four possibilities:
    - `mode = 0` → Player1: Human; Player2: Human
    - `mode = 1` → Player1: Human; Player2: Computer
    - `mode = 2` → Player1: Computer; Player2: Human
    - `mode = 3` → Player1: Computer; Player2: Computer

    Note: In this first version the strategy the Computer follows is trivial: It just consists on randomly generate a column to place the figure at.
5. **int status** → Game current state. There are five possibilities:
    - `status = 1` → Game is open. Player1 moves next.
    - `status = 2` → Game is open. Player2 moves next.
    - `status = 3` → Game is over. Player1 has won.
    - `status = 4` → Game is over. Player2 has won.
    - `status = 5` → Game is over. It finished with a draw.
6. **char** board** → Game board. It contains *r* rows and *c* columns.
    Each position takes one of these three values:
    - `board[i][j] = ' '` → Empty position.
    - `board[i][j] = 'X'` → Position occupied by Player1.
    - `board[i][j] = 'O'` → Position occupied by Player2.

The game is parameterised by the following five arguments:

1. *Connect*: Amount of pieces a player has to put consecutively in a row, column, left or right diagonal so as to win the game.
2. *Rows*: Amount of rows of the board.
3. *Columns*: Amount of columns of the board.
4. *Player1name*: Name of player1. If the name is "Computer", then the player is controlled automatically. Otherwise, the user is asked to control the player.
5. *Player2name*: Same as for player1.

3. Interactive Session:

The A02_Part1.zip file includes the file A02_Part1.exe with the application solved.

Compare your application to it so as to ensure you are achieving the desired functionality. Also, to become familiar with the application itself and its five parameters open a command line prompt and execute the program with the following configurations:

- A02_Part1.exe 4 6 7 Ruairi Nacho
  In this case, the standard version of Connect4 game is played.
  The board contains 6 rows and 7 columns.
  The two players (Ruairi and Nacho) are humans and thus controlled by the user. The first name (in this case Ruairi) is player1 and thus starts the game moving.
  The two players' goal is to win the game by placing 4 consecutive pieces ('X' and 'O', respectively) in a row, column or diagonal of the board.

- A02_Part1.exe 4 6 7 Computer Nacho
  In this case, the first player is named "Computer", so it is controlled by the machine.
  As mentioned before, for this first version of the program the strategy the "Computer" follows is trivial: It just randomly generates a column to place the figure at.

- A02_Part1.exe 4 6 7 Computer Computer
  In this case, both players are controlled by the machine.

- A02_Part1.exe 3 4 4 Nacho Ruairi
  In this case, a non-standard version of Connect4 game is played.
  The board only contains 4 rows and 4 columns.
  Also, now the two players (Ruairi and Nacho) try to win the game by placing 3 consecutive pieces ('X' and 'O', respectively) in a row, column or diagonal of the board.

4. Project:

The C project for this first part of the assignment consists just in 3 files:

1. game.h
2. game.c
3. main.c

game.h

This file is provided complete. It provides the following functionality:

0. Define: _CRT_SECURE_NO_WARNINGS
   This is useful so as to use the string functions of the library <string.h> we have seen at "Lecture 15-16: Strings". In particular, the functions strlen and strcpy are needed so as to copy the player names (passed as parameters to the program) to the fields p1 and p2 of the game variable. Moreover, the function strcmp is to be used so as to know if a player name is "Computer" or not, thus determining whether its movement is to be controlled by the user or by the machine.

0. Datatype: game.
   The type struct _game models the ConnectN game we want to play. As struct _game is a bit long, we type alias it to just game.

0. Datatype: bool.
   It allows you to use Booleans within your project, under the values False and True (please note the capital letters).

1. Function: game* create_new_game(char* p1, char* p2, int rows, int columns);
   It creates a new game provided the names of the two players and the rows and columns of the board.

2. Function: void display_board(char** board, int rows, int columns);
   It prints by console the current content of the board.

3. Function: void display_game_status(game g, int rows, int columns);
   It prints by console the content of the board and the status (next player to move if the game is still on; game result if the game is over).

4. Function: char my_get_char( );
   It reads one character entered by the user from the keyboard.

5. Function: int ask_for_column(int columns);
   It asks the user for an input until a valid column is entered.

6. Function: int user_movement(game* g, int columns);
7. Function: int computer_movement(game* g, int columns);
8. Function: void new_movement(game* g, int rows, int columns);

   These three functions are in charge of making a new movement. The current status of the game determines the player to move next, and its name determines whether it is a human or a computer player. Depending on it, function 6 or function 7 is to be applied.

9. Function: bool winning_row(char** board, int row, int ply, int columns, int connect);
10. Function: bool winning_column(char** board, int column, int ply, int rows, int connect);
11. Function: bool winning_diagonal(char** board, int diagonal, int ply, bool left, int rows, int columns, int connect);
12. Function: bool winning_player(game* g, int ply, int rows, int columns, int connect);

    Function 12 is responsible of controlling whether player ply (1 for player p1 and 2 for player p2) has won the game or not.
    To do so, the player must have placed connect consecutive pieces in either a row, a column, a left diagonal or a right diagonal. Functions 9, 10 and 11 isolate this possibility by evaluating a concrete row, a concrete column or a concrete diagonal.
    In particular, I found it handy to name the potential diagonals to be evaluated under the integer names 0, 1, 2, etc. However, if you prefer to call to winning_diagonal with the starting (x,y) position of the diagonal, then please feel free to change the body of the function.

13. Function: void update_status(game* g, int rows, int columns, int connect);
    This function is responsible of updating the status after each new movement.

14. Function: void play_game(int connect, int rows, int columns, char* p1, char* p2);
    This function is the main one. It creates a new game, and keep displaying the status and performing new movements while the game is not over.

game.c

Whereas functions 1 and 4 are provided complete, the rest of the functions are to be completed.

In function 1, please note that the two-dimensional array board is now defined in the heap, rather than in the stack. The code is provided complete, and it follows the same explanation given in the lectures.

main.c

This file is provided complete.

The int main(int argc, char* argv[]) just parses the 5 arguments from the program and calls to the function play_game. For debugging purposes, comment the lines parsing each of the parameters and force them to take concrete values. Once you have successfully programmed the application, you can uncomment the parameter parsing lines of code.

5. Evaluation and Submission Date:

- The full assignment is worth 25 marks. I will wait until the second part of the assignment is released for providing the mark breakdown.
- The completed file `game.c` has to be submitted as a single zip file to Blackboard before Sunday 27th November 11.59pm.
- A short interview which each student will take place in week 12 to discuss the code.