APS 105 — Computer Fundamentals
Lab #6: Functionality for the Connect6 Game
Fall 2013

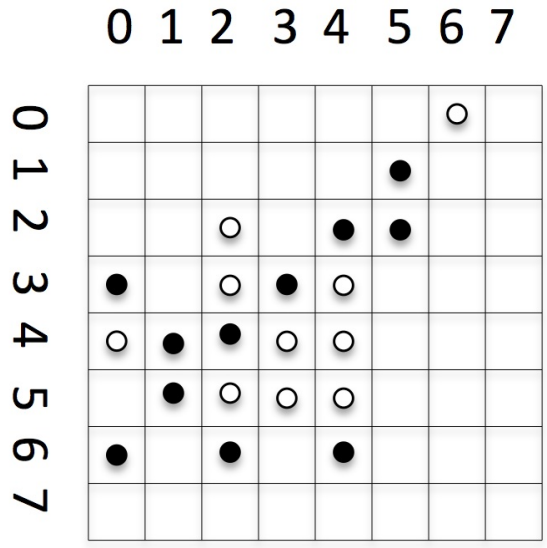
Important note: The course material necessary for completing this lab will be found in your lecture notes and in the Carter text up to the end of Chapter 7. The `malloc` and `free` functions are discussed in Section 10.2.

You must use the `submitaps105f` command to electronically submit your solution by 11:59 p.m. on **Saturday, November 9, 2013**.

Objective - Part of the Code for a Connect6 Game

The goal of this lab is to write a program that will be used (in Lab 7) as part of a *Connect6* game, as well as a little bit of the 'thinking' code that will be used in that lab to have the computer play against a human opponent.

Here is a brief description of the full Connect6 game, only some of which you'll be working on in this lab: Connect6 is played on a board (like a chess board or a checkers board) that has dimensions $n \times n$. In the picture below $n = 8$. One player uses white stones and the other black, and they take turns placing stones on the board. (Black, the first player places 1 stone, then White places two, and then Black places two and every player places two in a turn after that). The objective of the game is to be the first player to place six of their colour stone in a row – horizontally, vertically, or diagonally. In the picture below, black has won the game with six diagonal stones:



For this lab you will write a C language program that will do the following:

1. The first input to the program will be n, giving you the size of the $n \times n$ board. Your program *must* build a dynamically-allocated 2-dimensional array of `char` type variables of that size. You can assume that the value of n will be greater than 6 and less than 20.

2. The next sequence of inputs will describe *all* of the moves made by black and white players to a specific point in a game. This will be the locations, on the board, of

all of the black and white stones to some specific point in a game. Each location is described as a colour of stone (the letter 'B' or 'W') and a row, column location of the stone. The end of this sequence will be designated by using the letter 'E' as the colour. As illustrated in the figure above, position `0,0` is the top-left corner of the board. After *every* entry, the program should print out the current state of the board, your program will print out the state of the board, one letter for each square, as follows:

```
U - for unoccupied
B - occupied by black
W - occupied by white
```

For example, after the entire board above is entered, it would be printed as follows:

```
UUUUUUWU
UUUUUBUU
UUWUBBUU
BUWBWUUU
WBBWWUUU
UBWWWUUU
BUBUBUUU
UUUUUUUU
```

To print the board, your program should contain a function with the following prototype:

```
void printBoard(char **board, int n);
```

where `board` is the 2D array representing the current board state, and `n` is the board dimensions. Note that your program must detect and print an error if an attempt is made to place a stone at a square that is already occupied (see sample output below for an example of this).

3. Once the board has been input, the program should check the entire board to see if either black or white has won the game – if there are six in a row of white or black, either horizontally, vertically, or diagonally. If there is, then your program should either print: "Black wins." or "White wins.", or if neither, then "No winner so far." You can assume that there is at most one winning sequence on the board. See the sample output below for an example of the required functionality.

4. The next phase of the program is to accept a sequence of inputs in the form of: ROW COL, where the ROW COL should be an *occupied* square of the current board. Your program should determine the colour of the stone on the square and then find the longest horizontal, vertical or diagonal contiguous (side-by-side) sequence of stones of that same colour that intersects with the given square. To do this, you must implement a function with the following prototype that realizes the specified behaviour:

```
int findLongest(char **board, int n, int row, int col);
```

where `board` is the 2D array representing the state of the board, `n` is the board dimensions, and `row` and `col` represent the row and column of an occupied position on the board. The function returns the length of the longest sequence. You will use

this function extensively in Lab #7 when you write a program that decides where to place stones on the board. Your program should terminate when -1 -1 is entered as the ROW COL. See the sample output below for an example. Your program may assume that in this phase, the user always enters the row and column of an occupied square; no error handling is needed to check for the case of the user entering the row and column of an unoccupied square. **HINT:** You can also use the `findLongest` function to determine if there is a winner of the game for the previous step above!

5. The last step is to use `free` to de-allocate the memory for the board.

We strongly encourage you to break up your program into separate functions, and to carefully test each function before separately, before connecting it into the larger program.

Write your program in a file called `Lab6.c`

It is highly recommended that you document your program with comments. Your comments should include at least your full name and student number.
Here is a complete execution of the program where the board is configured such that black wins. Your output must conform to this format:

```
Enter board dimensions (n): 8
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 0 0
BUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 1 1
BUUUUUUU
UBUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 2 2
```

```
BUUUUUUU
UBUUUUUU
UUBUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 3 3
BUUUUUUU
UBUUUUUU
UUBUUUUU
UUUBUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 4 4
BUUUUUUU
UBUUUUUU
UUBUUUUU
UUUBUUUU
UUUUBUUU
UUUUUUUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 5 5
BUUUUUUU
UBUUUUUU
UUBUUUUU
UUUBUUUU
UUUUBUUU
UUUUUBUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): W
Enter position (ROW COL): 2 2
Position is taken, try another move.
BUUUUUUU
UBUUUUUU
UUBUUUUU
UUUBUUUU
UUUUBUUU
UUUUUBUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): W
```

```
Enter position (ROW COL): 3 1
BUUUUUUU
UBUUUUUU
UUBUUUUU
UWUBUUUU
UUUUBUUU
UUUUUBUU
UUUUUUUU
UUUUUUUU
Enter stone (B|W): E
Black wins.
Enter an occupied position to check for contiguous stones (ROW COL): 0 0
Number of contiguous stones: 6
Enter an occupied position to check for contiguous stones (ROW COL): 1 1
Number of contiguous stones: 6
Enter an occupied position to check for contiguous stones (ROW COL): 3 1
Number of contiguous stones: 1
Enter an occupied position to check for contiguous stones (ROW COL): 5 5
Number of contiguous stones: 6
Enter an occupied position to check for contiguous stones (ROW COL): -1 -1
```

Here is another complete execution of the program where there is no winner:

```
Enter board dimensions (n): 7
UUUUUUU
UUUUUUU
UUUUUUU
UUUUUUU
UUUUUUU
UUUUUUU
UUUUUUU
Enter stone (B|W): W
Enter position (ROW COL): 2 2
UUUUUUU
UUUUUUU
UUWUUUU
UUUUUUU
UUUUUUU
UUUUUUU
UUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 3 3
UUUUUUU
UUUUUUU
UUWUUUU
UUUBUUU
UUUUUUU
UUUUUUU
UUUUUUU
Enter stone (B|W): W
Enter position (ROW COL): 3 1
```

```
UUUUUUU
UUUUUUU
UUWUUUU
UWUBUUU
UUUUUUU
UUUUUUU
UUUUUUU
Enter stone (B|W): B
Enter position (ROW COL): 3 4
UUUUUUU
UUUUUUU
UUWUUUU
UWUBBUU
UUUUUUU
UUUUUUU
UUUUUUU
Enter stone (B|W): W
Enter position (ROW COL): 3 4
Position is taken, try another move.
UUUUUUU
UUUUUUU
UUWUUUU
UWUBBUU
UUUUUUU
UUUUUUU
UUUUUUU
Enter stone (B|W): E
No winner so far.
Enter an occupied position to check for contiguous stones (ROW COL): 3 4
Number of contiguous stones: 2
Enter an occupied position to check for contiguous stones (ROW COL): 3 1
Number of contiguous stones: 2
Enter an occupied position to check for contiguous stones (ROW COL): -1 -1
```

Marking

A marking program will be used to automatically mark your lab. The program will use **the last version** of the file `Lab6.c` that you submitted using the `submitaps105f` command (details below).

This lab will be marked out of 10.

Full marks are given if your function works correctly, fewer if not, and zero if your file cannot be compiled. Late submissions will result in a mark of 0 for the entire lab. The deadline will be strictly enforced, so avoid last minute submissions.

You can run a testing program, called `tester`, yourself to test the correctness of your solution. At the command line in your ECF account, run:

```
/share/copy/aps105/lab6/tester
```

in the same directory as your solution file. The testing program will use a number of test cases to test your solution, and report `success` if your solution produces output that is

identical to the expected output. Some of these test cases will be used by the marking program as well, but the marking program will also be using other test cases that are not included in the testing program to test the correctness of your program. This implies that even though you do not have access to the marking program, you will obtain at least partial marks if all of the test cases in the testing program report `success` with your solution.

If you are unable to complete all of the work required in this lab, you will still be able to get partial marks for some of the test cases.

Your mark for this lab will contribute 3% to your mark in the course.

What To Submit

When you have completed the lab, use the command

```
submitaps105f 6 Lab6.c
```

within the directory that contains your solution to submit your file. Make sure you name your file exactly as stated (including lower/upper case letters). Failure to do so will result in a mark of 0 being assigned. You may check the status of your submission using the command

```
submitaps105f -l 6
```

where `-l` is a hyphen followed by the letter '*ell*'.