MP7 Testing and Debugging Connect 4

Due on Wednesday, October 22nd, 2014 at 10:00 pm

Overview

Your task this week is to test and debug the given code that implements the game Connect 4. There are a few bugs in the code that you have to find and fix. You need to test the code in a systematic way to locate and correct all the bugs using gdb. In the programming studio this week, we will show you how to debug a simple C code using gdb.

Connect 4 is a game where two players take turns choosing a column to drop either one X or O. The first one to connect 4 of the pattern wins. An example of the game screen is illustrated below:

```
   1 2 3 4 5 6 7
  ---------------
 | * * * * * * * |
 | * * * * * * * |
 | * O * * * * * |
 | O X * * * * * |
 | X X X X * * * |
 | X O O O * * * |
  ---------------

Player 1 Wins!!!
```

The Pieces

In the MP7 folder, you will find the following files:

`connect4.c` – This file contains the function definitions for the six functions explained in the Details section below. There are bugs only in this file.
`connect4.h` – This file contains the function prototypes for the six functions.
`main.c` – This file contains the main function which sets up the game board, gets the moves from the players, and checks for win using the functions defined in `connect4.c`.
`solution` – This is the executable file for the game with no bugs.

Details

```
int playAgain(char *str, char *again);
```

At the end of the game, the user is asked to play again. This function is used by the main function to check whether the input given by the user is a valid input, and if it is, the input is stored. You may not assume that the user will always input a valid string.

```
void getMove(int *col1, int *col2, int *col3, int *col4, int *col5, int *col6,
       int *col7, int *move, int playernum);
```

This function is used in the main function to get the move from the player. It uses `checkMove` to check whether it is a valid move. Again, you may not assume that the user will always input a valid string (or a valid integer).

```
int checkMove(int *col1, int *col2, int *col3, int *col4, int *col5, int *col6,
       int *col7, char *str, int *move);
```

The `checkMove` function is called by the `getMove`, and it is used to check whether the move is valid. It distinguishes whether the input from the user is a valid move and a valid input.

```
void makeMove(int *col1, int *col2, int *col3, int *col4, int *col5, int *col6,
       int *col7, int move, char XorO);
```

This function takes the move from the user, which is already validated by the `checkMove`, and modifies the board accordingly.

```
void checkWin(int *col1, int *col2, int *col3, int *col4, int *col5, int *col6,
       int *col7, int *win, char XorO);
```

Finally, the checkWin function goes through the board to find if any player has won. This is called each time after the players make the move.

Specifics

- Your code must be written in C and must be contained in the connect4.c file provided to you. We will NOT grade files with any other name.
- You must debug the functions in connect4.c.
- Your routine's return values and outputs must be correct.
- Your code must be well commented. You may use either C-style (/* can span multiple lines */) or C++-style (// comment to end of line) comments, as you prefer. Follow the commenting style of the code examples provided in class and in the textbook.
- You should work on your own for this MP. Partners are not allowed.

Building and Testing

Along with the codes, we will provide you with an executable that has no bugs. You may try running this to see how the program should be like. This file is called `solution`, as listed in The Pieces section. To run the `solution`, type:

```
./solution
```

To compile your MP, use the following command:

```
gcc -Wall -g main.c connect4.c -o mp7
```

To run your MP, type:

```
./mp7
```

To run your MP with gdb, type:

```
gdb ./mp7
```

For more gdb commands, click here.

Grading Rubric

Functionality (80%)
80% - All bugs are fixed. Points will be taken off depending on the number of bugs unfixed.

Style (10%)
5% - Compilation generates no warnings. Any warning means 0.
5% - Indentation and variable names are appropriate and reasonably meaningful.

Comments, clarity, and write-up (10%)
5% - Introductory paragraph explaining what you did. Even if it is just the required work.
5% - Code is clear and well commented.