Debugging Exercise - Battleship

Assignment Due: Friday, May 31 @ 11:59 pm

<table>
<thead>
<tr>
<th>Assignment Overview</th>
<th>Getting Started</th>
<th>Example Input</th>
<th>Detailed Overview</th>
</tr>
</thead>
<tbody>
<tr>
<td>C Modules</td>
<td>Assembly Modules</td>
<td>README File</td>
<td>Turnin Summary</td>
</tr>
</tbody>
</table>

Assignment Overview

The purpose of this program is to play battleship from the command line! The main() function is written in C, and it will run the game using various helper functions.

We provide all of the code to you, but the code doesn't quite compile or work as it should. It is up to you to track down the bugs and fix them. There are a total of 9 bugs (2 compile-time bugs, 1 linkage error, and 6 run-time bugs) in the source code.

You are required to record ALL of the bugs we placed and the solution for each bug. We will provide you with one of the bugs. See the section on Debug README File for more details.

You DO NOT need to modify any file/function headers.

Keep in mind that each bug fix should only modify at most three (3) lines of code. These lines may be separated by other lines, but if you can logically consider them as part of the same error, they should be documented together. However, if you find yourself thinking of a fix that changes more than this number of lines, either the bug is not in those lines, or your fix can be written more concisely.

Another thing to note: if, while trying to fix a bug, you notice some code is definitely wrong but fixing it does not get rid of the bug -- make a note of your fix for later, but work on solving the current bug first. Since you need to document the effects of each bug, you should undo this early fix until you find the actual fix to the current bug you are encountering.

Grading

- **README: 70 points** - See README details below.
- **Correctness: 30 points**

**NOTE:** If what you turn in does not compile with given Makefile, you will receive 0 points for this assignment.

Getting Started

Follow these steps to acquire the starter files and prepare your Git repository.

Gathering Starter Files:
The first step is to gather all the appropriate files for this assignment. Connect to pi-cluster via ssh.

```bash
$ ssh cs30xyz@pi-cluster.ucsd.edu
```

Create and enter the debug working directory.
Copy the starter files from the public directory.

$ cp ~/../public/debugStarterFiles/* ~/debug/

Starter files provided:

<table>
<thead>
<tr>
<th>Makefile</th>
<th>myRem.s</th>
<th>placeShip.c</th>
</tr>
</thead>
<tbody>
<tr>
<td>debug.h</td>
<td>setupBoards.s</td>
<td>parseSetupMode.c</td>
</tr>
<tr>
<td>debugStrings.h</td>
<td>askForSetup.c</td>
<td>printBoard.c</td>
</tr>
<tr>
<td>utils.h</td>
<td>checkWin.c</td>
<td>setupPlayers.c</td>
</tr>
<tr>
<td>autogenerateBoard.s</td>
<td>checkMove.c</td>
<td>placeGuess.c</td>
</tr>
<tr>
<td>intervalContains.s</td>
<td>debug2.c</td>
<td>utils.c</td>
</tr>
<tr>
<td>myRand.s</td>
<td>parsePlay.c</td>
<td></td>
</tr>
</tbody>
</table>

Example Input

A sample stripped executable provided for you to try and compare your output against is available in the public directory. Note that you cannot copy it to your own directory; you can only run it using the following command (where you will also pass in the command line arguments):

$ ~/../public/debugtest

NOTE:

1. The output of your program MUST match exactly as it appears in the debugtest output. You need to pay attention to everything in the output, from the order of the error messages to the small things like extra newlines at the end (or beginning, or middle, or everywhere)!

2. **We are not giving you any sample outputs, instead you are provided some example inputs. You are responsible for trying out all functionality of the program; the list of example inputs is not exhaustive or complete. It is important that you fully understand how the program works and you test your final solution thoroughly against the executable.**

Example input that has normal output:

cs30xyz@pi-cluster-105: ~/../public/debugtest

Welcome to Mini-Battleship!

This is a 2-player game. Players will be given the opportunity to manually place battleship pieces on the board, or have them randomly placed.

There are 2 types of ships to be placed:
- [D] Destroyer (takes 2 spaces)
- [S] Submarine (takes 3 spaces)

The objective of the game is to guess the locations of other player's ships. Follow the instructions that proceed. The first player to guess them all wins!
Do you want to randomly generate both boards, or manually place the water transport carriers yourself? Enter auto for automated setup, or manual for manual entry: auto

Player 1's turn.

Player 1's guesses:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
-------------
```

Please enter a position to guess: 0,4
You've hit a ship!

Player 2's turn.

Player 2's guesses:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
-------------
```

Please enter a position to guess: 0 0
Missing middle comma.
Please enter a position to guess: 00
Missing middle comma.
Please enter a position to guess: 0,0,2
Given column is an invalid number.
Please enter a position to guess: 0,5
Invalid location, or you've already guessed this position.
Please enter a position to guess: 0,0
You missed!

Player 1's turn.

Player 1's guesses:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
-------------
```
Please enter a position to guess: 1,4
You've hit a ship!

Player 2's turn.
Player 2's guesses:

Please enter a position to guess: 0,0
Invalid location, or you've already guessed this position.
Please enter a position to guess: 1,1
You missed!

Player 1's turn.
Player 1's guesses:

Please enter a position to guess: 2,4
You've hit a ship!
You sank a SUBMARINE!

Player 2's turn.
Player 2's guesses:
Please enter a position to guess: **m2,2**
Given row is an invalid number.
Please enter a position to guess: **2,2m**
Given column is an invalid number.
Please enter a position to guess: **2,2**
You missed!

Player 1's turn.
Player 1's guesses:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| # # # ~ ~ |
| ~ ~ ~ ~ ~ |
-------------
```

Please enter a position to guess: **3,4**
You've hit a ship!

Player 2's turn.
Player 2's guesses:

```
-------------
| m ~ ~ ~ ~ |
| ~ m ~ ~ ~ |
| ~ ~ m ~ ~ |
| ~ ~ m ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
-------------
```

Please enter a position to guess: **(3,3)**
Given row is an invalid number.
Please enter a position to guess: **3,3**
You missed!

Player 1's turn.
Player 1's guesses:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| # # # # ~ |
-------------
```
Please enter a position to guess: 3,3
You've hit a ship!
You sank a DESTROYER!

PLAYER 1 WINS!

cs30xyz@pi-cluster-105:

Note: You may also find that just entering input such as , or 1, or ,1 works. This is fine and intended – the blank basically represents position 0.

Detailed Overview

The function prototypes for the various C and Assembly functions are as follows.

C routines:
void askForSetup(struct playerInfo * player);
long checkMove(long board[NUM_ROWS][NUM_COLS], long row, long col);
long checkWin(struct playerInfo * player);
int main(int argc, char * argv[]);
long parsePlay(char * buffer, long * rowIdx, long * colIdx);
long parseSetupMode();
void printBoard(char board[NUM_ROWS][NUM_COLS]);
void placeGuess(struct playerInfo * player, long row, long col);
void placeShip(struct playerInfo * player, long ship, long startX, long startY,
               char * errFlag, long orientation);
long setupPlayers(struct playerInfo * player1, struct playerInfo * player2);
void unline();
void unlines(unsigned int n);

Assembly routines:
void autogenerateBoard(struct playerInfo * player);
int intervalContains(int value, int start, int end);
int myRand();
int myRem(int dividend, int divisor);
long setupBoards(struct playerInfo * player1, struct playerInfo * player2);

Board:
The board used in the Battleship game is set up differently from what you might expect, with the x values representing columns and the y values representing rows (instead of the “normal” way). Consider the following board below:

-------------
| ~ m ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ # ~ ~ |
| ~ ~ ~ ~ ~ |
The # represents a hit at (2, 4), not (4, 2), and m is at (1, 0), instead of (0, 1).

**Testing:**
So you don't have to spend so long playing the game multiple times, if you play with auto, the boards for player1 and player2 will always be setup as follows (# indicates a location taken up by a ship):

player1:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| # ~ ~ ~ ~ |
| # # # # ~ |
-------------
```

player2:

```
-------------
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ ~ ~ |
| ~ ~ ~ # ~ |
| # # # # ~ |
-------------
```

In addition, we've provided input files autoFile and manualFile that let you redirect input to read in these files so you don't have to type inputs by hand. autoFile will run the game using the setup mode "auto", whereas manualFile will run the game on "manual" with some arbitrary choice of ship placement (see the file for exactly where the ships get placed). Both files will end the game by hitting the exact spots where the ships are placed, so that one player wins.

To use the files, redirect them on stdin when running the debug executable:

```
$ ./debug < autoFile
$ ./debug < manualFile
```

**C Modules**

Listed below are the descriptions of modules written in C.

**askForSetup.c**

```
long askForSetup(struct playerInfo * player);
```

This function prompts the user to set up the battleship board for player.

**Return Value:** 1 if successful, END_PROGRAM on EOF.
checkMove.c

long checkMove(long board[NUM_ROWS][NUM_COLS], long row, long col);

This function checks if \texttt{board[row][col]} is a valid position. It first checks if the indices are in range, and then checks if the position has been filled yet.

\textbf{Return Value:} 0 if the position is not valid, 1 if valid.

checkWin.c

long checkWin(struct playerInfo * player);

This function checks if all of player's guesses have hit all of the ships on the enemy's board. \textbf{DO NOT CHANGE THIS FILE - there aren't any bugs here.}

\textbf{Return Value:} 0 if player hasn't won, 1 if player has won.

debug2.c

int main(int argc, char * argv[]);

The main driver of this program. It combines all of the game's logic as well as controlling the players and prompting them. It also contains the main game loop.

\textbf{Reasons for error:}
- Player enters EOF.

\textbf{Return Value:} \texttt{EXIT_FAILURE} if EOF was entered in \texttt{setupPlayers}, otherwise \texttt{EXIT_SUCCESS}.

parsePlay.c

long parsePlay(char * buffer, long * rowIdx, long * colIdx);

This function parses \texttt{buffer} (the user input from stdin). The input must be of the form \texttt{X, Y}. This function uses \texttt{strchr()} and \texttt{strtol()} to parse the X and Y arguments one at a time. It also replaces the comma and new line with null characters to allow \texttt{strtol()} to parse X and Y. It then stores the parsed X and Y values in \texttt{rowIdx} and \texttt{colIdx} respectively.

\textbf{Reasons for error:}
- Input isn't of the form \texttt{X, Y}.

\textbf{Return Value:} 1 if the user input string was successfully parsed, or 0 on failure.

parseSetupMode.c

long parseSetupMode();

This function parses the user inputted setup mode.

\textbf{Reasons for error:}
- Input mode isn't valid
Return Value: -1 if error, else the appropriate mode, or END_PROGRAM on EOF.

placeGuess.c
void placeGuess(struct playerInfo * player, long row, long col);

This function places a guess onto the player’s guess board. DO NOT CHANGE THIS FILE - there aren’t any bugs here.

placeShip.c
void placeShip(struct playerInfo * player, long ship, long startX, long startY,
                char * errFlag, long orientation);

This function places a ship on a player’s board. If an error occurs in placing the ship, it sets errFlag accordingly and returns.

Reasons for error:
● Ship is out of bounds horizontally or vertically, or would collide with another ship if placed.

printBoard.c
void printBoard(char board[NUM_ROWS][NUM_COLS]);

This function prints out the given board.

setupPlayers.c
long setupPlayers(struct playerInfo * player1, struct playerInfo * player2);

This function sets up the two player’s boards, clearing the boards with memset(), initializing struct playerInfo for both players, and prompting the user for a mode.

Return Value: Returns the user-inputted mode.

utils.c
void unline();
void unlines(unsigned int n);

This file defines some utility functions to use in the mini battleship game. DO NOT CHANGE THIS FILE - there aren’t any bugs here.

Assembly Modules

Listed below are the descriptions of modules written in ARM assembly.

autogenerateBoard.s
void autogenerateBoard(struct playerInfo * player);
Automatically (randomly) places all ships onto the player's board. If the random location does not work for placing the ship, then it will continue finding other locations until it can.

intervalContains.s

```c
int intervalContains(int value, int start, int end);
```

Determines whether or not `value` is inside the interval `[start, end]`. This interval is inclusive on both ends. Keep in mind the parameters are different from PA1’s `intervalContains`!

**Reasons for error:**
- If `start` is greater than `end`, return -1 for error.

**Return Value:**  -1 if the interval is invalid, 1 if `value` is contained in the interval, 0 if `value` is not contained in the interval.

myRand.s

```c
int myRand();
```

Generates a random byte (the algorithm for generating a random byte isn’t buggy).

**Return Value:**  A random byte.

myRem.s

```c
int myRem(int dividend, int divisor);
```

Calculates the remainder when dividing `dividend` by `divisor`. This should have the same behavior as the `%` operator in C.

**Reasons for error:**
- `divisor` is zero → result is undefined (we will not be checking for divide by 0)

**Return Value:**  The remainder.

setupBoards.s

```c
long setupBoards(struct playerInfo * player1, struct playerInfo * player2);
```

Sets up both players' boards based on the game mode. If the mode is AUTO, boards are auto-generated, and if the mode is MANUAL, boards are manually generated.

**Reasons for error:**
- user enters an invalid mode → return 0

**Return Value:**  1 if successful, 0 if an invalid mode is entered.
For the debugging assignments only, you do not have to include the usual high level description, how tested, etc. in your README file. You will, however, have to list the compilation error you encountered and the fix you made to correct the error.

You will also have to solve several logic errors. For each problem, describe:

- **Effects** of the bug. What signaled to you that this bug exists? This can be error messages for compiler errors, weird behavior for runtime errors, etc.
- The **line number(s)** of the fix(es). (These may change as you fix more bugs, so it's fine if the numbers aren't exact)
- What the line(s) looked like before and after your fix(es).
- A short explanation (1-2 sentences is fine) of your reasoning behind each fix and how you debugged the problem.

As a guideline, there should be 2 compilation errors/warnings, 1 linkage error, and 6 functionality problems. Make sure you locate all of them! (Note: When we say there is 1 compilation error, we mean that there is one fix you'll have to make, not that there is one error printed to the screen. In general, you should not have to modify more than 3 lines of code per fix).

1 functionality problem's location and effects are given to you below. On top of the bullet points mentioned above, you also need to explain specifically why the program crashed with this bug. Hint: it may be helpful to draw stack diagrams!

**Note:** You should attempt to fix this bug after you find all the other bugs! That way, you can follow along with the instructions below.

The final functionality bug is located at the lines: autogenerateBoard.s:66 and autogenerateBoard.s:110-111. It deals with the automated setup of both players’ boards.

Let's walk through how the program behaves because of this bug (note that your memory addresses may be different). Running the program with gdb:

$ gdb debug

Since the functionality bug pertains to the automated setup of the board, we can find the bug by first triggering that setup mode (see bolded text for input):

(gdb) r
Starting program: /home/linux/ieng6/cs30x/cs30xyz/debug/debug

Welcome to Mini-Battleship!

This is a 2-player game. Players will be given the opportunity to manually place battleship pieces on the board, or have them randomly placed.

There are 2 types of ships to be placed:
- [D] Destroyer (takes 2 spaces)
- [S] Submarine (takes 3 spaces)

The objective of the game is to guess the locations of other player's ships. Follow the instructions that proceed. The first player to guess them all wins!
Do you want to randomly generate both boards, or manually place the water transport carriers yourself? Enter auto for automated setup, or manual for manual entry: auto

or

(gdb) r < autoFile

Either way, you should end up with the following:
Program received signal SIGSEGV, Segmentation fault.
0x76fbd11c in memset () from /usr/lib/arm-linux-gnueabihf/libarmmem.so
(gdb)

Performing a backtrace reveals the following:
(gdb) bt
#0 0x76fbd11c in memset () from /usr/lib/arm-linux-gnueabihf/libarmmem.so
#1 0x00011918 in setupPlayers (player1=0x7efffa14, player2=0x7efff988) at setupPlayers.c:37
#2 0x00010c2c in main (argc=1, argv=0x7efffc04) at debug.c:63
(gdb)

Let's set a breakpoint at setupPlayers and re-run.
(gdb) break setupPlayers
Breakpoint 1 at 0x118b8: file setupPlayers.c, line 31.
(gdb) r

We hit our breakpoint (note that this is before the automated setup). Let's execute setupPlayers.c:31.
Breakpoint 1, setupPlayers (player1=0x7efffa14, player2=0x7efff988) at setupPlayers.c:31
31 memset(player1->board, WATER,
(gdb) next
33 memset(player1->guess, WATER,
(gdb) next
35 memset(player2->board, WATER,
(gdb)

Interestingly, there is no segmentation fault. Is GDB wrong? Let's type continue to run the program until we hit the segmentation fault. You'll need to type in auto to signal automated setup (if you redirected from the file, you won't have to input auto).
(gdb) c
Continuing.
Do you want to randomly generate both boards, or manually place the water transport carriers yourself? Enter auto for automated setup, or manual for manual entry: auto

Program received signal SIGSEGV, Segmentation fault.
0x76fbd11c in memset () from /usr/lib/arm-linux-gnueabihf/libarmmem.so
(gdb) bt
If for some reason you didn’t hit the segmentation fault immediately, type in `next` or `continue` until you do.

That’s strange: `setupPlayers` got called again? But we only called it once in our program: in `debug.c:63`.

To get full points on fixing this functionality bug, please include the aforementioned 4 parts (effect, line number of bug, fix, and a short explanation of why you made the fixes you did), as well as the answer to the following question: Why did this happen, or more specifically, why did the function `setupPlayers` get called again?

If you happen to find more than 9 errors, document all of them anyway. It is most likely that you are considering one error as many different errors. We will grade you based on what fixes you make - not the number of errors you find. In other words, you will not lose points for documenting extra errors.

## Turnin Summary

See the turn-in instructions [here](#). Your file names must match the below *exactly* otherwise our Makefile will not find your files.

### Final Turnin:

**Due:** Friday night, May 31 @ 11:59 pm

### Files required for the Final Turn-in:

- autogenerateBoard.s
- intervalContains.s
- myRand.s
- myRem.s
- setupBoards.s

- askForSetup.c
- checkMove.c
- checkWin.c
- debug2.c

- parsePlay.c
- parseSetupMode.c
- printBoard.c
- placeGuess.c
- placeShip.c
- setupPlayers.c

- debug.h
- debugStrings.h
- utils.h
- Makefile
- README

If there is anything in these procedures which needs clarifying, please feel free to ask any tutor, the instructor, or post on the Piazza Discussion Board. **DO NOT ask the tutors for help in the labs for this assignment – it is up to you to figure out how to fix it.**