The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2020
CSSE2310 / CSSE7231 - Assignment 1

Due: 18:00pm 23rd March, 2020
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision: 1.0

Integrity

This is an individual assignment. You should feel free to discuss aspects of C programming and the assignment specification with fellow students. You should not actively help (or seek help from) other students with the actual coding of your assignment solution. It is cheating to look at another student's code and it is cheating to allow your code to be seen or shared in printed or electronic form. You should note that all submitted code may be subject to automated checks for plagiarism and collusion. If we detect plagiarism or collusion, formal misconduct proceedings will be initiated against you. A likely penalty for a first offence would be a mark of 0 for the assignment. Don't risk it! If you're having trouble, seek help from a member of the teaching staff. Don't be tempted to copy another student's code.

Do not commit any code to your repository unless it is your own work or it was given to you by teaching staff. **If you have questions about this, please ask.**

While you are permitted to use sample code supplied by teaching staff this year in this course. Code supplied for other courses or other offerings of this course is off limits — it may be deemed to be without academic merit and removed from your program before testing.

Introduction

Your task is to write a C99 program (called `push2310`) which allows the user to play a game (described later). This will require I/O from both the user and from files. Your assignment submission must comply with the C style guide (v2.0.4) available on the course website.

This assignment is to be your own individual work. Using code which you did not write[1] is against course rules and may lead to a misconduct charge.

Help in Pracs

You can ask basic spec interpretation questions. For anything more complex, you will need to be signed off for both the linux and C tutes before getting help with the assignment. That is, `2310tool` must show "tute_help_for_ass1 Yes".

You can always ask for help with the C and linux tutes. You can always ask assignment questions on piazza.

Game instructions

The game is played on an $R \times C$ grid of cells (where R is the number of rows and C is the number of columns). The corners of the board are removed. Each cell has a point value. The value of the border cells is zero, interior cells have values between 1 and 9 (inclusive). For example, a 4×6 board would be drawn as:

[1] See the first lecture for exceptions

```
  0.0.0.0.
0.4.2.1.1.0.
0.9.1.1.5.0.
  0.0.0.0.
```

Empty cells are indicated with a dot. Players take turns placing "stones" on empty cells. The game ends when the interior of the board is full. For example:

```
  0.0.0.0.
0.4X2X1O1X0.
0.9O1O1X5O0.
  0.0.0.0.
```

Each player gets points for each cell they have a stone in.

- Player **O** $= 1 + 9 + 1 + 5$
- Player **X** $= 4 + 2 + 1 + 1$

Playing a stone on one of the edges, is different. Such stones will be pushed into the interior moving other stones as needed. This may result in a stone being pushed into the opposite edge. For example, Player X playing a stone at the ?

```
  0?0.0.0.
0.4X2.1O1.0.
0.9O1.1.5.0.
  0.0.0.0.
```

would lead to:

```
  0.0.0.0.
0.4X2.1O1.0.
0.9X1.1.5.0.
  0O0.0.0.
```

Player O playing at ?

```
  0.0.0.0.
0.4X2X1O1.0.
0?9X1.1O5.0.
  0O0.0.0.
```

would lead to

```
  0.0.0.0.
0.4X2X1O1.0.
0.9O1X1O5.0.
  0O0.0.0.
```

Stones can only be played in an edge cell if:

1. There is an empty cell in the direction it would be pushed.
2. There is a stone to be pushed immediately next to the edge cell

For example, in the following board:

```
  0A0B0.0.
0.4X2.1.1X0.
0D9O1O1X5.0.
  0O0.0C0.
```

stones could be played at C or D because there is space to push into and there is a stone immediately next to them (in the direction they would be pushing). Cell A would be invalid because there is no space to push into. Cell B would be invalid because there is a space next to it.

Scoring

A player's score at the end of the game is the total of the points for all cells which have that player's stones in them. For example:

```
  0.0.0.0.
0.4X2X1O1X0.
0.9O1X1O5O0.
  0O0.0.0.
```

Player X would have a score of $4 + 2 + 1 + 1 = 8$ points. Player O would have a score of $1 + 9 + 1 + 1 = 12$ points.

Invocation

The `push2310` program takes the following parameters in order:

1. The type of player for **O** — This must be one of `0, 1, H`. Where zero and one are automated players and H is a human player.
2. The type of player for **X** — same possibilities as above.
3. The save game file to load.

eg: `push2310 0 0 prev` would start game loading from a save file called `prev`. Both players will be type 0 automated players.

Note that even new games start with a save game.

Save game format

The first line of the file contains two space separated[2] integers (the height and width respectively). The second line contains a single character (either O or X) indicating which player is to have the next turn. The remaining lines are the contents of the board. Each cell is represented by two chars, a point value (between 0 and 9 inclusive) followed by one of **.**, **O**, **X** (with **.** being an empty cell). All lines are `'\n'` terminated and there should not be any extra lines in the file. For example:

```
6 6
X
  0.0.0.0.
0.2.1.1.1.0.
0.1.1.2.1.0.
```

[2] exactly one space

```
0.1.1.1.1.0.
0.2.1.1.2.0.
  0.0.0.0.
```

A board which has zero points in the interior or non-zero points in the border cells, is invalid. Both board dimensions must both be ≥ 3. Note that the top and bottom rows end in two trailing spaces (so all grid rows in the file are the same length).

Interaction

After the game has been loaded, and after each player makes a move, display the board. Whenever a human player needs to provide a move, display the following prompt:
?`:(R C)>`
where, *?* is either **O** or **X** (depending whose turn it is). Note that there is a single space character following the `>`. If the input is not valid, show the prompt again.

When an automated player makes a move, print the following before redisplaying the board:
`Player ?  placed at ?  ?`
where the missing values are:

1. The player character
2. The row the player placed in
3. The column the player placed in

At the end of the game, print (to `stdout`):
`Winners: O`
or
`Winners: X`
or
`Winners: O X`
as appropriate.

Saving games

When a human player is prompted, the game can be saved to a file by giving the input **s** followed by the name of the file. If saving fails, send the following message to `stderr`: `Save failed`

Whether the save succeeded or not, reprompt for an actual move.

Automated player behaviour

Type zero players only place stones in the interior of the board. If Player **O** is type zero, they will start in the top left (coordinates 1,1) and search right until they find an empty cell and place there. If they do not find a space in row 1, then they will start again at the beginning of row 2 etc. If Player **X** is type zero, they behave in a similar way, but starting at the bottom right and searching left (moving up a row as needed).

For example, on the following board:

```
   0.0.0.0.0.0.0.0.0.0.0.
0.2.1.1.5.1.1.1.1.6.1.1.1.0.
0.1.1.3.1.1.1.1.1.1.1.1.1.0.
```

```
0.2.1.1.1.1.2.1.1.1.1.1.7.0.
  0.0.0.0.0.0.0.0.0.0.0.0.
```

A few moves in, the board would look like:

```
  0.0.0.0.0.0.0.0.0.0.0.0.
0.20101050101.1.1.6.1.1.1.0.
0.1.1.3.1.1.1.1.1.1.1.1.0.
0.2.1.1.1.1.2.1X1X1X1X1X7X0.
  0.0.0.0.0.0.0.0.0.0.0.0.
```

The game would end, looking like:

```
  0.0.0.0.0.0.0.0.0.0.0.0.
0.201010501010101060101010O.
0.1010301010101X1X1X1X1X1X0.
0.2X1X1X1X1X2X1X1X1X1X1X7X0.
  0.0.0.0.0.0.0.0.0.0.0.0.
```

Type 1

To decide on type one moves, use the following in order (stop when a move is found):

- Check the edges for a move which would lower the other player's score. (check clockwise).
  - Row 0 from left to right
  - Rightmost column top to bottom
  - Bottom row, right to left
  - Column 0 bottom to top
- Find the empty interior cell with the highest value. In case of a tie, choose the first one found when searching left-to-right and top to bottom.

Exit status

Check for the error conditions in this table in the order given. For example, the named savefile does not exist and one of the player types is 'qq', then use error 2 not error 3.

Code	**Reason**	**Message** (to `stderr`)
0	Normal game over	
1	Incorrect number of args	`Usage: push2310 type0 typeX fname`
2	Player types are not `0,1,H`	`Invalid player type`
3	Can't read from file	`No file to load from`
4	Something wrong with the contents of save file	`Invalid file contents`
5	EOF on `stdin` when input required	`End of file`
6	The game being loaded has no empty cells in the interior	`Full board in load`

Submission

Submission must be made electronically by committing using subversion. In order to mark your assignment the markers will check out `https://source.eait.uq.edu.au/svn/csse2310-s???????/trunk/ass1`. Code checked in to any other part of your repository will not be marked.

Note: No late submissions will be accepted for this assignment (see ECP). The markers will evaluate the last commit in your repository before the deadline. Late by 1 minute (or less) is still late.

Compilation

Your code must compile with command:
`make`
When you compile, you must use at least the following flags: `-Wall -pedantic -std=c99`. You may add additional flags if you wish (eg `-g`) but you must not use flags or pragmas to try to disable or hide warnings.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. Clarifications may be issued via the the course discussion forum. Any such clarifications posted 5 days (120 hours) or more before the due date will form part of the assignment specification. If you find any inconsistencies or omissions, please notify the teaching staff.

Test Data

Testing that your assignment complies with this specification is your responsibility. Some test data and scripts for this assignment will be made available.
`testa1.sh` will test the code in the current directory. `reptesta1.sh` will test the code which you have in the repository. The idea is to help clarify some areas of the specification and to provide a basic sanity check of code which you have committed. They are not guaranteed to check all possible problems nor are they guaranteed to resemble the tests which will be used to mark your assignments.

Marks

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided that your code compiles (see above), you will earn functionality marks based on the number of features that your program correctly implements (as determined by automated testing), as outlined below. Partial marks may be awarded for partially meeting the functionality requirements[3]. Not all features are of equal difficulty. If your program does not allow a feature to be tested then you will receive 0 marks for that feature, even if you claim to have implemented it. For example, if your program can never open a file, we can not determine if your program would have loaded input from it. The markers will make no alterations to your code (other than to remove code without academic merit). Your programs should not crash or lock up/loop indefinitely. Your programs should not take a long time to run.

[3] ie passing some but not all of the tests of a feature

Please note that some features referred to in the following scheme may be tested in other parts of the scheme. For example, if you can not display the initial board, you will fail a lot of tests. Students are advised to pay close attention to their handing of end of input situations.

- Arg checking
  - Errors not related to loading (3 marks)
  - Errors related to loading (3 marks)
- Load and display (4 marks)
- First move
  - Type zero player (4 marks)
  - Type one player (4 marks)
- Save games (4 marks)
- Last move and gameover (4 marks)
- Partial games
  - Type zero and H players (6 marks)
  - Type one and H players (4 marks)
- Complete games (6 marks)

Style (8 marks)

Style marks will be calculated as follows:
Let A be the number of style violations detected by simpatico plus the number of build warnings. Let H be the number of style violations detected by human markers. Let F be the functionality mark for your assignment. Let M_A be the automated style mark and M_H be the human mark.

- If $A > 10$, then your style mark will be zero and M_H will not be calculated.
- Otherwise, let $M_A = 4 \times 0.8^A$ and $M_H = M_A - 0.5 \times H$ your style mark S will be $M_A + \max\{0, M_H\}$.

Your total mark for the assignment will be $F + \min\{F, S\}$.

A maximum of 5 violations will be penalised for each broad guideline area. The broad guideline areas are Naming, Comments, Braces, Whitespace, Indentation, Line Length and Overall. For naming violations, the penalty will be one violation per offending name (not per use of the name) up to the maximum of five. You should pay particular attention to commenting so that others can understand your code. The marker's decision with respect to commenting violations is final — it is the marker who has to understand your code. To satisfy layout related guidelines, you may wish to consider the `indent(1)` tool. Your style mark can never be more than your functionality mark — this prevents the submission of well styled programs which don't meet at least a minimum level of required functionality.

Example Sessions

```
$ ./push2310 0 0 board3
  0.0.0.0.
0.1.1.1.1.0.
  0.0.0.0.
Player O placed at 1 1
  0.0.0.0.
0.1O1.1.1.0.
  0.0.0.0.
Player X placed at 1 4
  0.0.0.0.
0.1O1.1.1X0.
  0.0.0.0.
Player O placed at 1 2
  0.0.0.0.
0.1O1O1.1X0.
  0.0.0.0.
Player X placed at 1 3
  0.0.0.0.
0.1O1O1X1X0.
  0.0.0.0.
Winners: O X
```

```
$ ./push2310 1 0 board3
  0.0.0.0.
0.1.1.1.1.0.
  0.0.0.0.
Player O placed at 1 1
  0.0.0.0.
0.1O1.1.1.0.
  0.0.0.0.
Player X placed at 1 4
  0.0.0.0.
0.1O1.1.1X0.
  0.0.0.0.
Player O placed at 0 4
  0.0.0.0.
0.1O1.1.1O0.
  0.0.0.0X
Player X placed at 1 3
  0.0.0.0.
0.1O1.1X1O0.
  0.0.0.0X
Player O placed at 0 3
  0.0.0.0.
0.1O1.1O1O0.
  0.0.0X0X
Player X placed at 1 2
  0.0.0.0.
0.1O1X1O1O0.
  0.0.0X0X
Winners: O
```

```
./push2310 1 H board7
  0.0.0.
0.1.1.1.0.
0.1.2.1.0.
0.3.1.3.0.
  0.0.0.
X:(R C)> 2 2
  0.0.0.
0.1.1.1.0.
0.1.2X1.0.
0.3.1.3.0.
  0.0.0.
Player O placed at 3 1
  0.0.0.
0.1.1.1.0.
0.1.2X1.0.
0.3O1.3.0.
  0.0.0.
X:(R C)> 2 1
  0.0.0.
0.1.1.1.0.
0.1X2X1.0.
0.3O1.3.0.
  0.0.0.
Player O placed at 3 3
  0.0.0.
0.1.1.1.0.
0.1X2X1.0.
0.3O1.3OO.
  0.0.0.
X:(R C)> 4 3
  0.0.0.
0.1.1.1.0.
0.1X2X1OO.
0.3O1.3X0.
  0.0.0.
Player O placed at 2 4
  0.0.0.
0.1.1.1.0.
0X1X2O1OO.
0.3O1.3X0.
  0.0.0.
X:(R C)> 4 3
  0.0.0.
0.1.1.1OO.
0X1X2O1X0.
0.3O1.3X0.
  0.0.0.
Player O placed at 0 3
  0.0.0.
0.1.1.1OO.
0X1X2O1OO.
0.3O1.3X0.
  0.0.0X
X:(R C)> 1 1
  0.0.0.
0.1X1.1OO.
0X1X2O1OO.
0.3O1.3X0.
  0.0.0X
Player O placed at 3 4
  0.0.0.
0.1X1.1OO.
0X1X2O1OO.
0.3O1X3OO.
  0.0.0X
X:(R C)> 0 1
  0.0.0.
0.1X1.1OO.
0X1X2O1OO.
0.3X1X3OO.
  OOO.0X
Player O placed at 3 4
  0.0.0.
0.1X1.1OO.
0X1X2O1OO.
0X3X1O3OO.
  OOO.0X
X:(R C)> 2 2
X:(R C)> 1 3
X:(R C)> 1 2
  0.0.0.
0.1X1X1OO.
0X1X2O1OO.
0X3X1O3OO.
  OOO.0X
Winners: O
```

Tips

1. No valid line of move input will contain more than 80 characters.
2. The internal representation of the board (how your program stores it) and the external representation do not need to be the same.