The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2015
CSSE2310 / CSSE7231 - Assignment 1
Due: 11:10pm 21 August, 2015
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 1.0

Introduction

Your task is to write an ANSI-C program (called `boxes`) 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 (v2.0.2) available on the course website.

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. You should read and understand the statements on student misconduct in the course profile and on the school web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-including-plagiarism`

In this course we will use the subversion (svn) system to deal with assignment submissions. 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.**

The Game

`boxes` will display a grid of cells (the corners indicated by `+`). The edges bordering the cells may be open (left blank) or closed (using `|` or `-` characters as appropriate). If all four edges of a cell are closed, then the cell may store a non-space character. For example:

```
+ + + +
  |
+ +-+-+
  |A|B|
+ +-+-+
|
+-+ + +
```

Beginning with an empty grid, players will take turns choosing an open edge to close. If the new edge means that a cell is now completely surrounded ("completed"), two things will happen: First, the player's symbol is placed in the cell. Second, the player has another turn immediately.

The game ends when there are no more open edges left in the grid or if EOF is detected when user input is required. The winners are the players with the most cells with their symbol in them.

The first player will use `A` as their symbol, second will use `B` and so on using successive ASCII values. Players will indicate their choice of edges using the coordinate of the upper/leftmost corner in the edge followed by `h` (horizontal edge) or `v` (vertical edge). For example, the edge in this grid:

```
+ + + +
    |
+ + + +

+ + + +

+ + + +
```

would be described by: 0 2 v (an edge starting at row 0, column 2 and going downwards).
Regarding extra turns:

1. A player will keep getting extra turns as long as their each extra turn completes a cell.
2. It is possible that a single edge will complete two cells, in this case the player only gets one extra turn for both cells.
3. A player does not get an extra turn if the game is finished.

Interaction

Before each player's turn, the current grid will be displayed. Each time the grid is printed, it will be followed by a newline. The user will be prompted with their symbol followed by `>`, then a space. For example:
`B>`
If the input describes a valid open edge, then the grid will be updated and the next turn begins (or the game ends). If the input is not valid, then the prompt will be reprinted for the same player to enter a different move. The program should keep prompting until a valid move is entered (or EOF).

If a valid move was entered, but the game is over, the (updated) grid will be printed one more time. Note that leading whitespace is not permitted in valid moves.

Example game

Note: extra gaps have been introduced between turns for clarity. In your implementation, there will be no blank lines between entering a valid move and the updated grid.

```
+ + +
+ + +
+ + +
A> 0 0 h

+-+ +
+ + +
+ + +
B> 0 0 v

+-+ +
|
+ + +
+ + +
A> 0 2 v

+-+ +
|   |
+ + +
+ + +
B> 1 2 h
B> 1 2 v

+-+ +
|   |
+ + +
    |
+ + +
A> 1 1 h

+-+ +
|   |
+ +-+
    |
+ + +
B> 0 1 h

+-+-+
|   |
+ +-+
    |
+ + +
A> 0 1 v

+-+-+
| |A|
+ +-+
    |
+ + +
A> 1 0 h

+-+-+
|A|A|
+-+-+
|   |
+-+ +
A> 2 0 h

+-+-+
|A|A|
+-+-+
    |
+-+ +
B> 1 0 v

+-+-+
|A|A|
+-+-+
|   |
+-+ +
A> 2 1 h

+-+-+
|A|A|
+-+-+
|   |
+-+-+
B> 1 1 v

+-+-+
|A|A|
+-+-+
|B|B|
+-+-+
Winner(s): A, B
```

Invocation

When run with no arguments, `boxes` should print usage instructions to stderr:

```
Usage: boxes height width playercount [filename]
```

and exit (see the error table).

`height` and `width` (measured in cells) must be integers greater than 1 and less than $1,000$. `playercount` must be an integer greater than 1 and less than 101. If `filename` is present, it will be interpreted as a path to a readable file containing a grid state to attempt to load as the start of the game. If `filename` is not present, then the game should begin with Player `A`'s turn on an empty grid.

Input file format

The first line contains only a single integer between 1 and number of players inclusive. This indicates which player is next to play.

The next set of lines describe the state of the edges in the grid ('0' for an open edge and '1' for a closed edge). The first line describes the first row of horizontal edges (left to right). The second line describes the first row of vertical edges (left to right). The third line describes the second row of horizontal edges and so on. Note that there will be more vertical edges in a row than horizontal edges.

The next set of lines describe which symbols are in the cells. Each row describes one row of cells (left to right). Each cell is described by a single integer indicating the player who completed the cell (0 indicates a cell which hasn't been completed yet). The integers in each row are separated by commas.

For example, a two player game on the board shown one Page 1 would look like:

```
1
000
0100
011
0111
011
1000
100
0,0,0
0,1,2
0,0,0
```

Faulty input files

Things to watch out for:

- Files with less lines than the grid size requires.
- Files with short lines (less content than expected).
- Files with lines which are longer than expected.
- Files with characters in the wrong places.
- Numbers outside the number of players.

Saving Games

There is another type of valid move. A player can enter, 'w' a space and then the path to the file to save the current game state in. The file should be output in exactly the same format described above for input files. After the save operation has been completed (or failed), your program should print the appropriate message (see table below) and then prompt for another move by the same player.

Errors and messages

When one of the conditions in the following table happens, the program should print the error message and exit with the specified status. All error messages in this program should be sent to standard error and followed by a newline. Error conditions should be tested in the order given in the table.

Condition	**Exit Status**	**Message**
Program started with incorrect number of arguments	1	Usage: boxes height width playercount [filename]
Invalid grid dimensions	2	Invalid grid dimensions
Invalid player count	3	Invalid player count
Cannot open grid file	4	Invalid grid file
Error reading grid. Eg: bad chars in input, not enough lines, short lines	5	Error reading grid contents
End of file while waiting for user input	6	End of user input
Misc system call failure	9	System call failure

For a normal exit, the status will be 0 and no special message is printed.

There are a number of conditions which should cause messages to be displayed but which should not immediately terminate the program. These messages should also go to standard error.

Condition	Action	Message
Error opening file for saving grid	Prompt again	Can not open file for write
Save of grid successful	Prompt again	Save complete

Compilation

Your code must compile with command:
`gcc -Wall -ansi -pedantic ass1.c -o boxes`

You must not use flags or pragmas to try to disable or hide warnings.

If any errors result from the compile command (ie the executable cannot be created), then you will receive 0 marks for functionality. Any code without academic merit will be removed from your program before compilation is attempted (and if compilation fails, you will receive 0 marks for functionality). If your code produces warnings (as opposed to errors), then you will lose style marks (see later).

Your solution must not invoke other programs or use non-standard headers/libraries. It must consist of a single, properly commented and indented C file called `ass1.c`.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. Clarifications may be issued via the the course newsgroup. 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. 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.

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 that no submissions can be made more than 72 hours past the deadline under any circumstances.

Test scripts will be provided to test the code on the trunk. Students are strongly advised to make use of this facility after committing.

Late Penalties

Late penalties will apply as outlined in the course profile. Remember, late penalties are determined automatically based on svn commit times. Late by 1 minute (or less) is still late.

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 your program correctly implements (as determined by automated testing), as outlined below. Partial marks will be awarded for partially meeting the functionality requirements. 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 complete.

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 grid, you will fail a lot of tests. Not just the one which refers to "initial grid". Students are advised to pay close attention to their handing of end of input situations.

- Command args — correct response to
  - incorrect number of args (1 mark)
  - invalid dimensions (2 marks)
  - invalid player count (2 marks)
  - invalid grid filename (2 marks)
  - errors in grid file (5 marks)
- Correctly display initial grid and prompt (2 marks)
- Reject illegal moves on the initial grid (2 marks)
- Correctly process a single move (2 marks)
- Correctly load grid and prompt (4 marks)
- Detect end of game (2 marks)
- Correctly save grid (4 marks)
- Play complete games (14 marks)

Style (8 marks)

If g is the number of style guide violations and w is the number of compilation warnings, your style mark will be the minimum of your functionality mark and:

$$8 \times 0.9^{g+w}$$

The number of compilation warnings will be the total number of distinct warning lines reported during the compilation process described above. The number of style guide violations refers to the number of violations of version 2.0.2 of the C Programming Style Guide. 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.

Notes and tips

1. A well written program should gracefully handle any input it is given. We should not be able to cause your program to `crash`.
2. Remember that the functionality of your program will be marked by comparing its output for certain inputs against the expected output. Your program's output must match exactly.
3. Be sure to handle unexpected end of file properly. Many marking tests will rely on this working.
4. Debug using small grids if possible.
5. Do not hardcode grid dimensions.
6. This assignment only deals with single byte characters (ie. characters which are stored in the `char` data type).

Updates

1.0

- Fixed a missing)
- Added an additional exit condition to cover system call failures which could not reasonably be predicted. The main culprit here would be malloc failure. We will not test that situation, but for those of you that want to check for it, there is an exit status and message for you now.

0.9.4

- Numbers in commandline arguments are to be interpretted using the standard C mechanisms. So could contain leading zeroes etc.

0.9.3

- To clarify acceptable input forms: Numbers for player moves shall be integers expressed in base 10. There shall be no leading or trailing spaces (except where necessary to act as a separator). There shall be no leading zeros or leading signs (+/−). No valid line will contain more than 30 characters.

  All of the above only restricts valid lines.

0.9.2

- Fixed the usage error message.

0.9.1

1. Clarified that width and height refer to the number of cells in each row and column respectively.
2. Just because a file exists to load a grid from, does not mean that file contains a valid grid. Your assignments need to be able to handle invalid files.
3. Fixed page number the example input refers to.