

The University of Queensland
School of Information Technology and Electrical Engineering
Semester One, 2011
COMP2303 / COMP7306 - Assignment 1
Due: 11pm 25 March, 2011
Marks: 50
Weighting: 25% of your overall assignment mark (COMP2303)
Revision 1.3

Introduction

Your task is to write a program (called `flip`) which plays a game using the rules given in this specification. This will require I/O from both the user and from files. Your assignment submission must comply with the C style guide 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 website: <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

The game is played by two players (**O** and **X**) on a square grid of cells. The cells are referred to by their row followed by their column (with each value beginning at *zero*). At the beginning of the game each player has two tiles in the centre of the board (see next page). If the board has an odd side length n , the top left centre tile will be at row and column $\lfloor \frac{n-1}{2} \rfloor$.

A player's turn consists of putting a tile in an empty cell. There must be a straight line between the new tile and one of the player's other tiles. There must be at least one of the opponents tiles between those tiles and no empty spaces. Any such tiles will now be owned by the player. Examples of moves (legal and otherwise) are shown on the next page.

If there are no legal moves for a player, then they must pass. If both players pass in consecutive turns, then the game ends.

Starting layout

```
+-----+
|.....|
|.....|
|.....|
|...OX...|
|...XO...|
|.....|
|.....|
|.....|
+-----+
```

Legal first moves for **O** shown as **o**.

```
+-----+
|.....|
|.....|
|...o...|
|...OXo...|
|...oXO...|
|...o...|
|.....|
|.....|
+-----+
```

Top option chosen

```
+-----+
|.....|
|.....|
|...o...|
|...OX...|
|...XO...|
|.....|
|.....|
|.....|
+-----+
```

In between tile flipped

```
+-----+
|.....|
|.....|
|...O...|
|...OO...|
|...XO...|
|.....|
|.....|
|.....|
+-----+
```

For this board with legal moves for **O** shown as 'o'/'!':

```
+-----+
|.....|
|.o.!Xo..|
|.X.XXX..|
|OOOOXOO.|
|.X.XXX..|
|.o.o.o..|
|.....|
|.....|
+-----+
```

Placing in 1 3 forms two lines and results in

```
+-----+
|.....|
|...OX...|
|.X.OOX...|
|OOOOXOO.|
|.X.XXX..|
|.....|
|.....|
|.....|
+-----+
```

Illegal moves

Illegal moves for **X**. None of the 1's enclose an **O** tile. Legal moves are indicated with **x**. Since any legal move will touch another tile, the cells marked with a dot are not legal either.

```
+-----+
|.....|
|...1...|
|...xOx..|
|..1001..|
|..1XOx..|
|..1111..|
|.....|
|.....|
+-----+
```

Moves such as 2 4 are illegal because the cell is already occupied.

Specification

The program (`flip`) can be run in three ways:

- When run with no arguments it should print usage instructions:

```
Usage: flip load filename
       or flip new dim [playerXtype] [playerOtype]
```

Note that “or” is indented four spaces.

- Loading a game from a file. A previously saved game is resumed.
- Begin a new game with a `dim×dim` board. If no other parameters are given (or they are 0), the game will have two human players. Player types of 1 or 2 represent computer controlled players. For example: `flip new 8 1` would give a game where **O** is controlled by a human and **X** is controlled by the computer.

When the game begins, the board should be printed. When a move is required from a human player, a prompt should be printed.

Player (O)>

or

Player (X)>

If the player gives input which is not a valid move, the prompt should be reprinted. Once a valid move is entered, the move should be performed, the board displayed. If the game is not over, a prompt should be displayed for the next player.

For a computer player’s turn, no prompt is displayed, instead their move is displayed once it is made. If there are no valid moves for a player, then no prompt is displayed and the board is not reprinted. Instead, a message is displayed and it is the other player’s turn again.

```
+-----+
|X.X.O.X.|
|XX0000X!|
|X.X.X.X.|
|...XXXX.|
|.XXXXX..|
|XXX.....|
|.X.....|
|.....|
+-----+
Player 0 moves at 1 7.
+-----+
|X.X.O.X.|
|XX000000|
|X.X.X.X.|
|...XXXX.|
|.XXXXX..|
|XXX.....|
|.X.....|
|.....|
```

```
+-----+
X passes.
Player 0 moves at 3 1.
```

When the game ends, a message is output giving the tile counts for each player.

```
+-----+
|XXXXXXXXO|
|XXXOXXOO|
|XOXXOXOO|
|XXXXO000|
|XOXXO000|
|XOXXOXOX|
|XXXO00XX|
|XXXXOXXX|
+-----+
Game Over - O=24 X=40.
```

If the game requires input from standard in but none is available (eof) then a message is displayed indicating this and which player was involved.

```
Player (0)> End of input from Player 0.
```

Compilation

Your code must compile with command:

```
gcc -Wall -ansi -pedantic flip.c -o flip
```

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 `flip.c`.

Player types

- 0 a "human" player. Moves are read from standard input. Moves should be entered as row followed by column (space separated). The top left corner of the board will be 0 0.
- 1 a computer player. Start at the top left corner and move along the row until a valid move is found. If there are no valid moves in the first row start again at the left of the second row and so on.
- 2 a computer player. Same as 1 except start in the bottom right corner and search right to left. Move up one row if there are no valid moves in the bottom row.

Saving games

A human player can choose to save the game when it is their turn. They do this by giving `s` *immediately* followed by a filename. (possibly referring to a different directory). The game exits when this is successful.

```
./flip new 4
+-----+
|....|
|.OX.|
|.XO.|
|....|
+-----+
Player (0)> s/wontwork
Unable to write to /wontwork.
Player (0)> smygames/newgame
Game saved.
```

Note that the first attempt failed because the user is not allowed to create files in the root directory.

Save game files can be used with the `load` parameter for the program. These save game files will also need to store the player types. When the game is loaded, the board should be displayed and the game continue from the point where the player saved.

This specification does not describe a format for saved games. Instead, loading will be tested using files which *your program* has saved. As such, if your assignment does not implement save functionality properly, then it will not be able to load properly either.

Errors

All messages in this program should be sent to *standard out*.

In the following table, ? represents a value to be filled in by the program. Unless otherwise specified all messages are followed by a newline. Where '-' appears, do not exit.

Condition	Exit Status	Message
Program is started with 0 params	1	<i>Display usage instructions</i>
Game over (Board is full)	0	Game Over - O=? X=?.
Game over (Board not full)	0	? passes. ? passes. Game Over - O=? X=?.
Game saved	0	Game saved.
Board size is not an integer greater than 3	2	Invalid board dimension.
Player type is not 0 or 1 or 2	3	Invalid player type.
Unable to load from the specified file	4	Error loading board.
Unable to save game	-	Unable to write to ?.
s command given with no filename	-	Please give a filename.
End of input before end of game	5	End of input from Player ?.
Program is started with other invalid combination of params	1	<i>Display usage instructions</i>

Style

You must follow *version 1.5* of the COMP2303/COMP7306 C programming style guide found at http://www.itee.uq.edu.au/~comp2303/resources/c_resources.html.

Submission

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

The due date for this assignment is 25th March at 11pm.

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

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 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 save a file, we can not determine if your program would have loaded a game successfully. 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.

- Command line errors (8 marks)
 - Does the program respond correctly to bad command line parameters?
- Human players (14 marks)
 - First move (6 marks)
 - Complete game (8 marks)
- Computer players (12 marks)
 - Do they place tiles correctly?
 - Do they attempt invalid moves? (they shouldn't)
 - Are the appropriate messages displayed?
 - Can they play a complete game correctly?
- Load/Save (both human and computer players) (8 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 1.5 of the COMP2303/COMP7306 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.

Late Penalties

Late penalties will apply as outlined in the course profile.

Specification Updates

It is possible that this specification contains errors or inconsistencies or missing information. It is possible that clarifications will be issued via the course website. 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

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.* Testing that your assignment complies with this specification is still *your* responsibility.

Notes and tips

- Where possible debug on a small board.
- Do not attempt to write complex operations in one go. Decompose them into simpler tasks and get those working first.
- The ability to check for lines of opponent's tiles between two cells (without modifying the board) is critical to this assignment.
 - Do not write separate functions to handle each possible direction.
 - See your tutors for strategies.
- Make sure your code is generic — the size of the current board should not be hardcoded.
- Possible save strategies
 - A “snapshot” of the current state of the board.
 - Record of all the moves in the game so far.
 - See your tutors for pros/cons of these.
- GET STARTED!

Revisions

- 1.2 → 1.3
 - Valid inputs: For command line parameters, the whole string for the parameter must be checked. So 1.2 and 1bob are not valid. For move inputs, if “%d %d” would scan two values then it is considered valid. So 1 2.4 and 3 1tree would be valid.
 - Which error message to give? Check for errors in the order given in the table and use the first one which is applicable.
- 1.1 → 1.2
 - Fixed Legal move mistake on page 2.
 - Removed references to `make`.
- 1.0 → 1.1
 - Fixed missing `.`
 - Player **O** always moves first.
 - Clarified usage message.
 - Relationship between prompting for input and end of file: If you read at least one character of input then you should display another prompt.

Sample Session

```
./flip new 4
+-----+
|. . . .|
|.OX.|
|.XO.|
|. . . .|
+-----+
Player (0)> 0 2
+-----+
|..0.|
|.00.|
|.XO.|
|. . . .|
+-----+
Player (X)> 0 3
+-----+
|..OX|
|.OX.|
|.XO.|
|. . . .|
+-----+
Player (0)> 3 1
+-----+
|..OX|
|.OX.|
|.00.|
|.0..|
+-----+
Player (X)> 1 3
Player (X)> help
Player (X)> 3 0
+-----+
|..OX|
|.OX.|
|.XO.|
|XO..|
+-----+
Player (0)> 2 0
+-----+
|..OX|
|.OX.|
|000.|
|XO..|
+-----+
Player (X)> 1 0
```

```
+-----+
|. .OX|
|XXX.|
|XOO.|
|XO..|
+-----+
Player (0)> 0 1
+-----+
|.00X|
|XOX.|
|XOO.|
|XO..|
+-----+
Player (X)> 3 2
+-----+
|.00X|
|XOX.|
|XXX.|
|XXX.|
+-----+
Player (0)> 0 0
Player (0)> 1 3
+-----+
|.00X|
|X000|
|XXX.|
|XXX.|
+-----+
Player (X)> 0 0
+-----+
|XXXX|
|XX00|
|XXX.|
|XXX.|
+-----+
0 passes.
Player (X)> 2 3
+-----+
|XXXX|
|XXXX|
|XXX.|
+-----+
0 passes.
X passes.
Game Over - 0=0 X=15.
```