The University of Queensland
School of Information Technology and Electrical Engineering
Semester One, 2012
CSSE2310 / CSSE7231 - Assignment 1

Due: 11pm 23 March, 2012

Marks: 50

Weighting: 25% of your overall assignment mark (CSSE2310)

Revision 1.2

Introduction

Your task is to write a program (called `noline`) 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 web-site: `http://www.itee.uq.edu.au/itee-student-misconduct-i`

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). The players take turns placing tiles on empty cells [all cells are empty at the start of the game]. Player O makes the first move. The player who forms a line (horizontal, vertical or diagonal) of three or more of their own adjacent tiles **loses**.

The game ends in a draw if there are no empty cells left but neither player has lost.

Example game

```
-----
.....
.....
.....
.....
.....
=====
O> 0 0
-----
O....
.....
.....
.....
.....
=====
X> 0 4
-----
O...X
.....
.....
.....
.....
=====
O> 4 4
-----
O...X
.....
.....
.....
....O
=====
X> 3 1
-----
O...X
.....
.....
.X...
....O
=====
O> 3 0
-----
O...X
.....
.....
OX...
....O
=====
X> 1 2
-----
O...X
..X..
.....
OX...
....O
=====
O> 2 4
-----
O...X
..X..
....O
OX...
....O
=====
X> 4 0
-----
O...X
..X..
....O
OX...
X...O
=====
O> 2 1
-----
O...X
..X..
.O..O
OX...
X...O
=====
X> 1 4
-----
O...X
..X.X
.O..O
OX...
X...O
=====
O> 4 1
-----
O...X
..X.X
.O..O
OX...
XO..O
=====

X> 3 3
```

```
-----
O...X
..X.X
.O..O
OX.X.
XO..O
=====
O> 0 3
-----
O..OX
..X.X
.O..O
OX.X.
XO..O
=====
X> 1 0
-----
O..OX
X.X.X
.O..O
OX.X.
XO..O
=====
O> 2 0
-----
O..OX
X.X.X
OO..O
OX.X.
XO..O
=====
X> 4 2
-----
O..OX
X.X.X
OO..O
OX.X.
XOX.O
=====
O> 4 3
-----
O..OX
X.X.X
OO..O
OX.X.
XOXOO
=====
X> 3 4
-----
O..OX
X.X.X
OO..O
OX.XX
XOXOO
=====
O> 2 3
-----
O..OX
X.X.X
OO.OO
OX.XX
XOXOO
=====
X> 0 1
-----
OX.OX
X.X.X
OO.OO
OX.XX
XOXOO
=====
O> 0 2
-----
OXOOX
X.X.X
OO.OO
OX.XX
XOXOO
=====
X> -10 0
X> 0 0
X> 0 5
X> 4
X> cats
X> 1 1
-----
OXOOX
XXX.X
OO.OO
OX.XX
XOXOO
=====
Player X loses.
```

Invocation

When run with no arguments it should print usage instructions to stderr:

```
Usage: noline dim [playerXtype [playerOtype [Oin Oc
```

When run with any number of arguments other than ~~0,~~ 1, 2, 3 or 7, the program should exit with an error (see table later).

The meaning of the parameters is as follows:

Parameter	**Meaning**
dim	Side length of the grid (must be an odd integer >1)
playerXtype	‘1’ or ‘2’ denotes a computer controlled player, ‘0’ denotes a “human” player
playerOtype	(as above) defaults to 0 if not specified.
Oin	Where to read playerO’s moves from: ’-’ use stdin; anything else will be treated as a filename.
Oout	Where to send messages/output for playerO to: ’-’ use stdout; anything else will be treated as a filename
Xin, Xout	(as above)

If the last four arguments are not given, then they default to ’-’.

Compilation

Your code must compile with command:

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

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 `noline.c`.

Player types

0 A "human" player, that is, moves are read in rather than generated by the program itself.

1 Instead of reading moves, this player type will check the cell (row, column) for a given n (where n=0 for first move, 1 for second move, . . .):
$k = n \cdot (s + 2) \bmod (s^2)$,

row= k/s, col= $k \bmod s$.
Note: s is the side length of the board.

2 Very similar to type 1 except that it starts in the opposite corner $(s-1, s-1)$ and counts backwards. For example: if s= 11, then the first cells to try would be:

   - 10 10
   - 9 8
   - 8 6
   - 7 4
   - ...

Player types 1 and 2 should print the move they chose to their output (followed by a newline).

Errors

All error messages in this program should be sent to standard error. Error conditions should be tested in the order given in the following table. All messages are followed by a newline.

Condition	**Exit Status**	**Message**
Program started with invalid number of arguments	1	*Display usage instructions*
Board dimension is not an odd integer >1	2	Invalid board dimension.
Player type is not 0, 1, 2	3	Invalid player type.
Can not access files given as args 4, 5, 6. 7	4	Invalid files.

Note that entering an invalid move is not an error condition. In such cases, the prompt for that player should be displayed again and more input read.

Other Messages

The following messages should be sent to both players. If both players are using stdout, then only print the message once.

Condition	**Messages**
Game over (Board is full)	The game is a draw.
Game over (player? loses)	Player ? loses.
Game over (EOF for player?)	Player ? loses due to EOF.

Move Sequence

When it is a player's turn:

1. Print the board (to their output).
2. Display prompt and wait for input.
3. If player enters an invalid move, go to 2.

End of Game Behaviour

When the game ends, print the board to both players. Then print the end of game message to both players. Note: If both players are using stdout then only print these items once.

Valid Input

Integer command arguments must not contain any other characters. For move input (which should be processed a line at a time), if scanf("%d %d", ...) would extract 2 integers from it correctly [and the line contains less than 80 characters] then it is acceptable. No line longer than 80 characters is considered valid.

Style

You must follow version 1.6 of the C programming style guide found at:
`http://courses.itee.uq.edu.au/csse2310/2012s1/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 Friday 23th March at 11pm. Note that no submissions can be made more than

120 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.

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 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.

- Command args

  - invalid dimension (1 mark)
  - usage instructions for incorrect number of args (1 mark)
  - player type invalid (1 mark)
  - file access errors (1 mark)
- Correctly display initial board (2 marks)
- Correctly handle initial move (4 marks)
- Complete games
  - "human" players only (10 marks)
  - "human" vs AI (6 marks)
  - AI vs AI (4 marks)
  - Large boards (6 marks)
  - Split input and output (6 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 1.6 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.

Test submissions

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.

  - 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.
- How you handle EOF is important since some of the tests rely on it.
- GET STARTED!

Updates

- If noline is run with no arguments, then print usage instructions and exit with 1.
- We will not test an AI player with an invalid input file (since AI players do not attempt to use the file).
- Clarified meaning of symbols in the move formulae.