The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2016
CSSE2310 / CSSE7231 - Assignment 1
Due: 11:10pm 19 August, 2016
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 1.0.2

Introduction

Your task is to write an C99 program (called `nogo`) which allows the user to play a game of atari-go(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

`nogo` will display a grid of cells like this:

```
/----\
|....|
|....|
|....|
|....|
\----/
```

The dots indicate empty cells (the other marks are just borders). The two players in the game will take turns putting "stones" into empty cells. The first player will indicate their stones with `O` and the second player will use `X`. The cells are numbered from the top left corner as 0,0 (row, then column).

Connected sets of stones of the same type form "strings". To be connected, stones need to be horizontally or vertically adjacent. In this example:

```
/----\
|OOO.|
|...O|
|.X..|
```

```
|X.X.|
\----/
```

The first player has 2 strings while the second player has 3.

Strings may have "liberties" (spaces they could grow into). A liberty is an empty cell horizontally or vertically adjacent to a string. In this example:

```
/----\
|X.OX|
|..OO|
|..X.|
|....|
\----/
```

The `X` at 0,3 has no liberties; the `X` at 2,2 has 3 liberties and the string of Os has 3.

A string is considered "captured" when it has no liberties. So in the example above, the string at 0,3 would be captured. The first player to have one of their strings captured loses. In cases where both players would have a string captured for example:

```
/-----\
|OX OX|
|OOXXX|
|O....|
|.....|
\-----/
```

the player placing the stone checks their opponent's strings first. So, if O placed at 0,2 then X would lose.

Interaction

At the beginning of the game and after each player's turn, the grid will be displayed. There are two types of player in the game, `h` (where moves will be read from `stdin`) and `c` (where moves will be generated by your program). For `c`-players, the move the program generates will be printed like this (substitute X for O as needed):
`Player O: 1 0`
then go to the next players turn. For `h`-players, the user will be prompted:
`Player X>`
and wait for the user to enter *row column* (single space separated with no leading nor trailing spaces). If the user's input is not valid (eg not two numbers, not within the bounds of the grid, not empty), then the prompt is reprinted for them to try again.

When a winner is determined, the following is displayed:

```
Player ? wins.
```

(substitute O/X for ? as appropriate).

Invocation

When run with an incorrect number of arguments, `nogo` should print usage instructions to stderr:

```
Usage: nogo p1type p2type [height width | filename]
```

and exit (see the error table).

`p1type` and `p2type` must be either `c` or `h`. `height` and `width` (measured in cells) must be integers between 4 and 1,000 inclusive. If height and width are omitted, then a filename can be given instead, in that case, the program should try to load a saved game from the named file.

Saved game file format

Saved game files will consist of a line of 9 numbers (separated by single spaces) followed by the contents of the grid (without borders). For example:

```
5 5 1 4 1 4 2 0 0
XX...
....O
.....
O....
O....
```

The numbers are (in order):

- height of the grid
- width of the grid
- 0 or 1 to indicate which player is next to play. 1 indicates X is to play next.
- the next two numbers indicate the row and column a computer player for `O` will try.
- the number of the next move for the computer player for O.
- the next two numbers are the next row and column for a computer player for `X` will try.
- the number of the next move for the computer player for X.

Note that the last number is zero, but X has made 2 moves already. This is because in this example, Player-X was a human and so did not use the computer player. 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.
- Next player something other than 0 or 1.

Computer player moves

The computer players will generate their moves using the following algorithm (note that the player should continue generating moves until an empty space is found):

- I_r is the initial row (1 for O and 2 for X).
- I_c is the initial column (4 for O and 10 for X).

- F is a multiplication factor (29 for O and 17 for X).
- G_w is the width of the grid.
- G_h is the height of the grid.
- M is a counter of moves generated (starting at zero).

Initial setup:
$B = I_r * G_w + I_c$
$r = I_r$, $c = I_c$
To get a move, return ($r \bmod G_h$) and ($c \bmod G_w$).
To generate the next move.

- if M is a multiple of 5:
  - $N = (B + M/5 * F) \bmod 1000003$
  - $r = N/G_w$ (integer division)
  - $c = N \bmod G_w$
- if $M \bmod 5$ is:
  1. $r+ = 1$, $c+ = 1$
  2. $r+ = 2$, $c+ = 1$
  3. $r+ = 1$, $c+ = 0$
  4. $r+ = 0$, $c+ = 1$

So for a 7×7 grid the move sequence generated for Player O would start with:
$(1, 4), (2, 5), (4, 6), (5, 6), (5, 0)$
$(5, 5), (6, 6), (1, 0), (2, 0), (2, 1)$
$(2, 6), (3, 0), (5, 1), \ldots$

Saving games

To save a game, instead of entering a row and column, enter `w` followed immediately by the path of the file to save to. That is, no space between `w` and the start of the path.

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: nogo p1type p2type [height width | filename]
Incorrect player types	2	Invalid player type
Board dimensions invalid	3	Invalid board dimension
Can't open save file for reading	4	Unable to open file
Error reading game. Eg: bad chars in input, not enough lines, short lines	5	Incorrect file contents
End of file while waiting for user input	6	End of input from user

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	Unable to save game

Compilation

Your code must compile with command:
`make`
When you compile, you must use at least the following flags: `-Wall -pedantic -std=gnu99`.

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.

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.
`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.

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 types (2 marks)
  - invalid load filename (2 marks)
  - errors in load 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 (3 marks)
- Play complete games with computer players (8 marks)
- Play complete games (7 marks)

Style (8 marks)

If g is the number of style guide violations and w is the number of compilation warnings, A is the number of style warnings as detected by the style checker, and F is your functionality mark: your style mark will be:

$$\begin{cases} = \min\{F,\ 8 \times 0.8^{g+w}\} : \text{if } A < 12 \\ = 0 : \text{if } A \geq 12 \end{cases}$$

That is, your style mark will be set to zero if you have more than eleven (capped) style violations.

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. You can assume that no valid line of input contains more than 70 characters.
3. 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.
4. Be sure to handle unexpected end of file properly. Many marking tests will rely on this working.
5. Debug using small grids if possible.
6. Do not hardcode grid dimensions.
7. You are expected to check for inability to open files, but do not need to consider other system call failures such as malloc fails.

Example game — two computer players

```
/----\
|....|
|....|
|....|
|....|
\----/
Player O: 1 0
/----\
|....|
|O...|
|....|
|....|
\----/
Player X: 2 2
/----\
|....|
|O...|
|..X.|
|....|
\----/
Player O: 2 1
/----\
|....|
|O...|
|.OX.|
|....|
\----/
Player X: 3 3
/----\
|....|
|O...|
|.OX.|
|...X|
\----/
Player O: 0 2
/----\
|..O.|
|O...|
|.OX.|
|...X|
\----/
Player X: 2 0
/----\
|..O.|
|O...|
|XOX.|
|...X|
\----/
Player O: 1 2
/----\
|..O.|
|O.O.|
|XOX.|
|...X|
\----/
Player X: 0 3
/----\
|..OX|
|O.O.|
|XOX.|
|...X|
\----/
Player O: 1 3
/----\
|..OX|
|O.OO|
|XOX.|
|...X|
\----/
Player O wins
```