The University of Queensland
School of Information Technology and Electrical Engineering
Semester Two, 2013
CSSE2310 / CSSE7231 - Assignment 1
Due: 11:10pm 16 August, 2013
Marks: 50
Weighting: 25% of your overall assignment mark (CSSE2310)
Revision 1.2

Introduction

Your task is to write an ANSI-C program (called `thebox`) which simulates firing a ball into a box of reflectors and other components to see where it comes out. This will require I/O from both the user and from files. Your assignment submission must comply with the C style guide (v1.7) 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 System

`thebox` will simulate a rectangular grid of cells. Each cell is occupied by a single component. The following table describes each component type, the character used to display it and the effect on a ball entering the cell. Throughout this specification, the compass directions North, South, East and West are abbreviated N,S,E,W respectively.

Component	**Char**	**Affect**
Space	.	Ball continues moving in the same direction
ReflectorNESW	/	Ball changes direction: $W \rightarrow S$, $N \rightarrow E$, $S \rightarrow W$, $E \rightarrow N$
ReflectorNWSE	\	Ball changes direction: $W \rightarrow N$, $N \rightarrow W$, $S \rightarrow E$, $E \rightarrow S$
Reflector	=	Ball reverses direction
Launchpad	@	Ball jumps over the next 4 cells in the current direction of movement, lands on the fifth (moving in the same direction). Note: the jump is instantaneous.
Teleport	A-Z	Ball's movement direction remains unchanged but it jumps to the cell marked with the next letter. If there are no more letters, then it jumps back to the first letter. Note: the jump is instantaneous. That is, in one step you enter the A cell and in the next step you leave the B cell.

The path of the ball through the box will be indicated with using digit characters. Start with '1' and use higher numbers where the path self-intersects (see the = example below). Note that there is no special character indicating the position of the ball. It is just the front of the path. Also note that the path is not drawn over objects. So, in the fourth frame of the / example, the ball is in the cell with the /.

In the @ example, the ball is on the @ in the 4th frame.

/ example

```
......
.../..
......
......

......
.../.1
......
......

......
.../11
......
......

......
.../11
......
......

......
.../11
...1..
......

......
.../11
...1..
...1..
```

\ example

```
......
...\..
......
......

......
...\.1
......
......

......
...\11
......
......

......
...\11
......
......

......
...1..
...\11
......
......
```

= example

```
......
...=..
......
......

......
...=.1
......
......

......
...=11
......
......

......
...=11
......
......

......
...=21
......
......
......

...=22
......
......
```

@ example

```
.........
......@..
.........
.........

.........
......@.1
.........
.........

.........
......@11
.........
.........

.........
......@11
.........
.........

.........
.1....@11
.........
.........
.........

.........
11....@11
.........
.........
```

Letter example

```
......
....A.
.C.B..
.D....

......
....A1
.C.B..
.D....

......
....A1
.C.B..
.D....

......
....A1
.C1B..
.D....

......
....A1
.C1B..
.D....

......
....A1
.C1B..
1D....
```

The Letter example may need some more explanation. In the third frame, the ball is on top of the 'A'. In the fourth frame, it leaves the 'B'. In the fifth frame, it is on the 'C'.

Invocation

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

```
Usage: thebox mapfile [maxsteps]
```

and exit (see the error table). Note that the [ ] indicate that the parameter is optional. The maximum steps must be strictly positive and less than 1000. If the maximum number of steps is not given, your program should

assume a value of 10.

When run with a correct number of arguments, `thebox` should read in the map file. It should then display the grid and then display the prompt:

```
(side pos)>
```

The user will then enter a letter indicating one of the sides of the grid followed immediately by a position along that side (starting at 1). If the user enters invalid input at this prompt, the program should display a new prompt and try again.

Once you have valid input, your program will then fire a ball into the grid from that position. In each step, the ball will move one grid cell (either horizontally or vertically). The ball will keep moving in the same direction unless it runs into a cell which changes its direction. After each step, the program will display the grid (followed by a blank line), with the path of the ball shown on it. The first time the ball enters an empty cell, that cell should be indicated with a 1. Each time the ball reenters that cell, the number should be incremented. Once the number reaches 9, do not increment it any further (this does not affect the path of the ball).

The program will keep processing steps until either:

1. The ball moves outside the grid.
2. The number of steps the ball has moved has exceeded the maximum given on the command line (check this after the ball has moved in each step).

When one of these conditions is met, the program will print the message:

```
End of simulation.
```

Then display the original grid and prompt for a new start location.

Map format

The first line of the map file will contain two strictly positive integers ($r\,c$) separated by a space (both less than 1000). These integers specify the number of rows and columns in the map (in that order). The remaining r lines in the file will each consist of c characters from the table above (each line should be terminated with a '\n').

Errors

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. Error conditions should be tested in the order given in the table (but after for a note on 5 and 6). All messages are followed by a newline.

Condition	Exit Status	Message
Program started with incorrect number of arguments	1	*Display usage instructions*
Invalid maxium number of steps	2	Bad max steps.
Cannot open map file	3	Missing map file.
Can't read valid map dimensions from file	4	Bad map dimensions.
Illegal character in map file	5	Bad map char.
Too much or too little data in the map file	6	Map file is the wrong size.
The map contains letters, but some between the smallest and largest are missing. For example: A,B,E or C,D,L. Or there is only one letter. Or there are duplicate letters	7	Missing letters.

When end of input on `stdin` is reached, exit the program with status 0 and no message.

Note on errors 5 and 6. These errors should be raised when they are first detected (if processing one character at a time). Too much or too little data happens when while moving along a line, a newline is encountered sooner than expected, or a (# or .). If as you walk along a line you see some other invalid character, that should trigger "Illegal character", even if the line would later turn out to have the wrong number of chars.

Compilation

Your code must compile with command:

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

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 `thebox.c`.

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. 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

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.

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 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. 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 inital grid, you will fail a lot of tests. Not just the one which refers to "initial grid".

- Command args
  - usage instructions for incorrect number of args (1 mark)
  - invalid max steps (1 mark)
  - can't open mapfile (1 mark)
  - invalid dimensions in file (1 mark)
  - illegal character in map file (1 mark)
  - Incorrect amount of data in the map file. (1 mark)
  - Invalid letter sequence (1 mark)
- Correctly display initial grid (4 marks)
- Correctly process a single shot into an empty grid. (4 marks)
- Correctly process a single shot into a grid using:

  - A single / (1 mark)
  - A single \ (1 mark)
  - A single = (1 mark)
  - A single @ (3 marks)
  - An A and a B (4 marks)
- Correctly process a number of shots into grids using only one type of component. (5 marks)
- Correctly process multiple shots into grids using multiple components.
  That is, process a sequence of single shots, one after the other. (12 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 1.7 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. A number of marking tests will rely on this working.
4. Debug using small grids if possible.
5. Do not hardcode grid dimensions. (If you do, your functionality mark will be capped at 20 marks. That is, you will not be able to get a functionality mark higher than 20).
6. Note: any time the grid is displayed, a blank line should be printed immediately afterwards.

Ammendments

- 1.1
    1. Emphasised that this is ansi-C.
    2. Added extra text for @
    3. Made (side pos)> prompt consistent.
    4. No leading or trailing blanks on valid input. [Don't focus on this though].
- 1.1ζ
    1. Fixed URL.
    2. Fixed example session.
- 1.2
    1. Clarified errors 5 and 6.

Example Session

```
./thebox small 4
...
.\.
...
...
...

(side pos)>E1
..1
.\.
...
...
...

.11
.\.
...
...
...

111
.\.
...
...
...

End of simulation.
...
.\.
...
...
...

(side pos)>N2
.1.
.\.
...
...
...

.1.
.\.      #Note that only blank
...      #cells show numbers
...
...

.1.
.\1
...
...
...

End of simulation.
...
.\.
...
...
...

(side pos)>N3
..1
.\.
...
...
...

..1
.\1
...
...
...

..1
.\1
..1
...
...

..1
.\1
..1
..1
...

End of simulation.
...
.\.
...
...
...

(side pos)> ^D
```