The University of Queensland
School of Information Technology and Electrical Engineering

Semester One, 2010

COMP2303 / COMP7306 – Assignment 1(v1.3)

Due: 11pm Monday March 29, 2010
Marks: 50
Weighting: 25% of your overall assignment mark (COMP2303)

Introduction

Your task is to write a program (called `naval`) which simulates one side of a naval battle. Ships will be positioned on a grid and the user must guess where they are. The ship positions will be read from a file specified on the command line. As well as developing your C programming skills, this assignment will test your ability to follow a specific coding style guide.

This is an **individual assignment**. You should feel free to discuss aspects of C programming and the assignment specification with your fellow students, but you shouldn't 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/about_ITEE/policies/student-misconduct.html

Specification

The program (`naval`) will take two filenames as parameters. The first file will describe the rules for the game. The second file will contain the locations of the ships.
To make things simpler to begin with, giving `standard.rules` as the rules file will use the following rules (even if there is no file called `standard.rules`):

- The grid will be 8 by 8 cells.
- There will be five ships in the following order:
  1. 5 cells long.
  2. 4 cells long
  3. 3 cells long
  4. 2 cells long
  5. 1 cells long

It is acceptable to supply a standard.rules file with your assignment provided that it matches the one given in this specification.

The file describing the locations of the ships (called the map file) will consist of lines of the following form (ending in \n only):
x y d
where x and y are positive integers (starting at 0) which give the column and row where the ship begins. The d is one of N, S, E, W indicating the direction the rest of the ship is relative to the start point. For example:

0 0 S
1 1 S
4 2 E
5 5 N
7 0 W

Would produce the following layout:

1							5
1	2						
1	2			3	3	3	
1	2						
1	2				4		
					4		

Ships are not permitted to overlap or cross outside the grid so positioning the second ship at 0 4 S would be invalid.
The format for the rules file is as follows (all lines end in \n):
The first line consists of two postitive integers (>0) giving the width and height of the grid.
The next line consists of a single integer giving the number of ships in a game. You may assume there will not be more than 15 ships.
The rest of the file consists of lines with a single integer describing the length of each ship.
So standard.rules would be written:
8 8
5
5
4
3
2
1

Assuming the file(s) are present (errors will be described later),
./naval standard.rules map1.map
should do the following:

i) Display the current board:
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .
. . . . . .

ii) Display the prompt:
(x,y)>
There is no space following the prompt.
iii) Wait for input of two integers indicating the guess of the location of a ship.
If the coordinates given match the location of a ship, then the program should print “Hit”. If the coordinates do no match then print “Miss”. If the input is too big, too small or invalid in some other way, then the program should print “Bad guess”. ~~For example, there should be no characters after the two numbers~~. Please see changes at the end of the document for more details about input errors.
iv) When the last cell of a ship has been hit, the program should print “Ship sunk”.
v) When the last ship has been sunk the program should print “Game over” and then exit.
vi) If the game is not over go to i)

For example:

```
(x,y)>2 0
Miss
../.....            #Note that misses are indicated by /
........
........
........
........
........
........
........
(x,y)>1 2
Hit
../.....
........
.*......            #Note that hits are indicated by *
........
........
........
........
........
(x,y)>7 0
Hit
Ship sunk
../....*
........
.*......
........
........
........
........
........
```

```
# Some time later
../....*
**......
**..***.
**......
**../*/.
/.../*/.
........
........
```
(x,y)>0 0
Hit
Ship sunk
Game over
The program should exit with status 0.

Please note that your program output must match this specification exactly. The functionality part of your assignment will be evaluated programmatically so deviation (even in matters like whitespace) is not acceptable. If your assignment cannot perform simple tasks it may not be tested for more complex functionality. For example if your program can read a map file but cannot display it we cannot determine if it has read the file correctly.

Your submission must include all source and any other required files (in particular you must submit a Makefile). Do not submit compiled files (eg .o or compiled programs)..

Your program must compile with:
`make`

Your program must be compiled under gcc with at least the following switches:
`-ansi -pedantic -Wall`
You are not permitted to disable warnings or use pragmas to hide them.

If any errors result from the make command (i.e. the `naval` executable can not be created) then you will receive 0 marks for functionality (see below). 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).

Your program must not invoke other programs or use non-standard headers/libraries.

Errors:

The table below describes how your program should respond to error conditions. The message should be printed to standard ~~error~~ out and then the program should exit with the status given. Under normal operation, your program ~~should not print anything and~~ should exit with status zero. There should not be any combination of input which causes your program to crash or loop indefinitely. (Except of course if the user repeatedly guesses the same cells).

Both files should be opened before either of them are processed (rules first then map). If the standard rules are in use (and you are not reading a file) pretend that it opened successfully. Extra parameters on the command line should be silently ignored.

Error	Message	Exit status
Not enough parameters	Usage: naval rules map	10
Rules file is missing or unreadable	Missing rules file	20
Map file is missing or unreadable	Missing map file	30
Error processing rules file	Error in rules file	40
Ships overlap in map	Overlap in map file	50
Ship goes out of bounds	Out of bounds in map file	51
Other error processing map file	Error in map file	52
Input runs out before game is over.	Bad guess	60

Style:

You must follow **version 1.4** of the COMP2303/COMP7306 C programming style guide found at http://www.itee.uq.edu.au/~comp2303/resources/c_resources.html.

Submission:

Your assignment submission must be committed to your subversion repository under ~~2010~~ass1/trunk. Your submission time for the assignment will be considered to be the time for the last commit in the 2010ass1/trunk directory.

You must ensure that all files needed to compile and use your assignment (including a *Makefile*) are committed and not just sitting in your working directory. Do not commit compiled files or binaries. I strongly suggest checking out a clean copy for testing purposes.

The due date for this assignment is 11pm Monday March 29, 2010. The policy on “grace days” and exceptional circumstances (e.g. illness) is as outlined in the course profile. Submissions completed

before 11pm Sunday March 28, 2010 will earn additional grace days. Note that no submissions can be made more than 120 hours past the deadline under any circumstances.

Marks:

Marks will be awarded for both functionality and style.

Functionality (42 marks)

Provided 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 can detect hits correctly. The markers will make no alterations to your code (other than to remove code without academic merit).

Marks will be assigned in the following categories.

1. Program correctly handles invalid command lines (2 marks)
2. With standard rules, ships horizontal (east) [16 total]
   - Correctly handle “hit” (2 marks)
   - Correctly handle “miss” (2 marks)
   - Correctly detect “sunk” (2 marks)
   - Correctly handle bad guess (1 mark)
   - Correctly play whole game (7 marks)
   - Correctly detect invalid maps (2 marks)
3. With standard rules, ships in any direction [16 total]
   same breakdown as #2
4. Non-standard rules [8 total]
   - detect invalid map (1 mark)
   - detect invalid rules (1 mark)
   - Correctly play whole game (6 marks)

Style (8 marks)

Your style mark will be the minimum of:

$8 * 0.9^{(\text{Number of style guide violations + number of compilation warnings})}$
and
your functionality mark.

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 1.4 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 if no grace days remain at the time of submission.

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.

Corrections:

[v1.3]

1. Replaced X with * to indicate hits. (This matches published test data).
2. Repeating a guess:
   If the player makes a guess they have already tried, the hit or miss message should be displayed as before. However hitting a ship which has already been sunk should **not repeat** the sunk message.
3. Your assignments must compile with -ansi -pedantic -Wall
4. Ships cannot have lengths <1. Number of ships must be >0.

[v1.2]

1. Errors to go to standard out not standard error.
2. Fixed svn submit url.
3. Fixed inconsistent text for bad guesses.
4. Changes to input errors:
   1. No line of input (in any file) should be longer than 20 chars. If a line is longer than this, it should produce an input error (bad guess/error in rules file/error in map file).
   2. If a line is not too long then extra chars after the valid input should be ignored.
   3. Regarding extra white space, any situation where scanf would ignore the extra

whitespace (within a line) is acceptable. For example: scanf("%d %d" ....
would accept 5 6 as well as 5 6

4. As before, if there are more lines of input than required the extra lines should be ignored.

[v1.1]

1. In any situation where an input source contains additional lines of input after the program would expect end of input, any extra lines of input should be silently ignored.
   If extra input occurs on the end of a line of input, then it should be silently ignored.
2. Any messages printed should be followed immediately by a newline.
3. Fixed an error in the map example.
4. Forced map size to be >0
5. Fixed inconsistancy in Ship sunk text.
6. Other minor fixes in red.