COSC2138/CPT220: Programming Principles 2A
Study Period 2, 2015

Assignment 1: Hangman and Minesweeper Games

Due date – Sunday 12th July 2015, 11:59pm

Abstract

In this assignment, you are asked to implement two separate programs simulating a the hangman game and the minesweeper game to demonstrate early knowledge of C programming principles. The concepts covered include:

- Data types and basic operators
- Branching and looping programming constructs
- Basic input and output functions
- Strings
- Functions
- Pointers
- Arrays
- Basic modularisation

Please note that you must read the supplementary document “COSC2138/CPT 220 Programming Principles 2A/- General Assignment Information” in the Assignment Section of Blackboard   prior to reading this assignment, a document that presents important general information pertaining to all assignments including the three general requirements, Requirements 6-9.

You are expected to have a partly modularized solution using all functions in the start-up code provided as a minimum.

Permission to change the start-up code is not normally given, unless there is very good reason (i.e.: a bug). You will need to build upon the code that is provided, instead. But note that you can add your own functions as described in **Requirement 5.**

If you have any concerns about the start-up code, please post your query to the **Blackboard Assignment 1 Discussion Forum.**

Functional Requirements

This section describes in detail all functional requirements of assignment #1. A rough indication of the number of marks for each requirement is provided. You should make a conscientious effort to write your program in a way that is capable of replicating the examples given in these requirements. Start with the simplest requirements first.

Section 1: Hangman game

For this program, you need to implement a hangman game simulation for the user. A word is chosen randomly from a predefined list of words, and the user guesses letters one at a time. The game is over when either the user makes 10 wrong guesses, or the user has successfully guessed all letters and identified the word.

Here are couple of websites showing how the game is played:
http://en.wikipedia.org/wiki/Hangman_(game)

For other online demo, have a look here:
http://www.playhangman.com/

Requirement #1 – Implementation of Hangman start up code - 25 marks

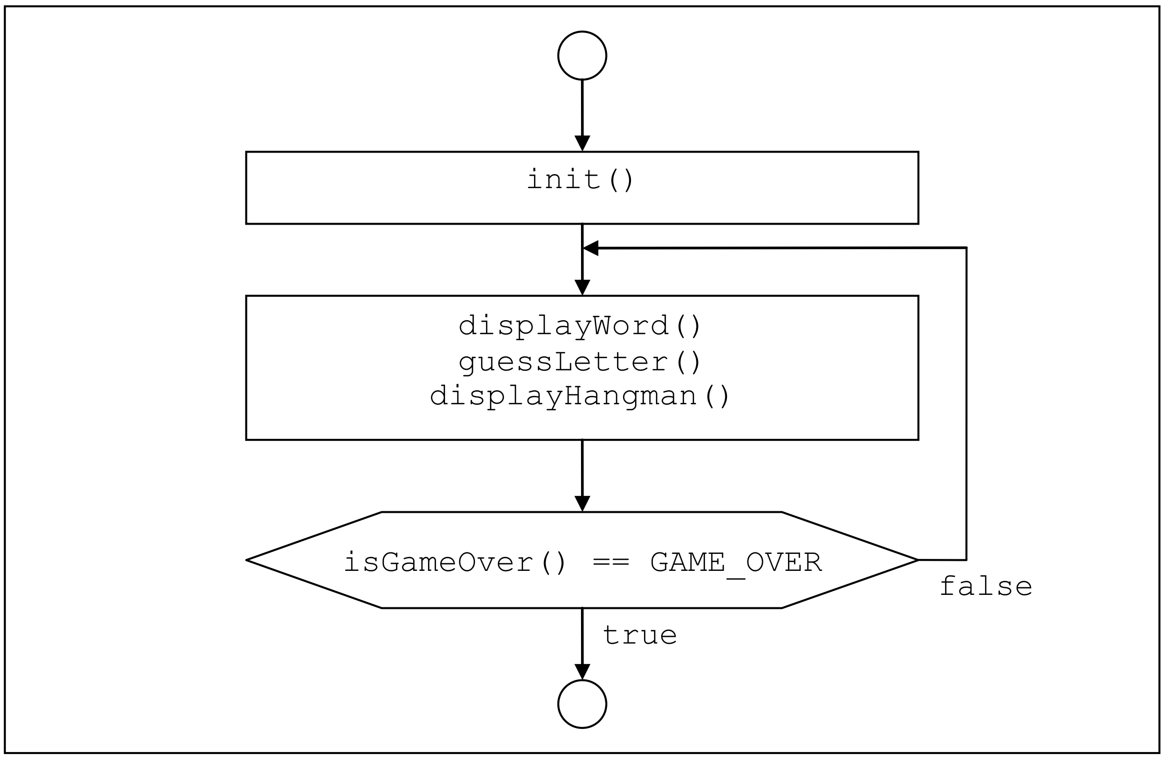
The start up code for the hangman game has empty function definitions for the following five functions. It is your job to implement them according to the descriptions in the start up code.

- `init()`
- `displayWord()`
- `guessLetter()`
- `displayHangman()`
- `isGameOver()`

You can get full marks for each of these only if your program runs them correctly. These are worth 5 marks each.

Requirement #2 – Program flow and execution - 15 marks

To gain full marks for this requirement, you will need to fully implement your `main()` function so that it runs your hangman simulation according to the flow chart over the page.

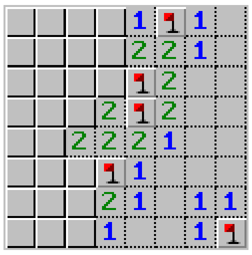


Section 2: Minesweeper game

For this program, you need to implement a text-menu driven version of the popular Minesweeper game. Minesweeper is a simple game provided with most default Windows installations.

Minesweeper is a popular game available on most operating systems. The goal of the game is to find where all the mines are located within an **`M x N`** field

Below is an example of a partially solved Minesweeper puzzle with 64 squares. This is the GUI (Graphical User Interface) version, and your text-menu driven version will look somewhat similar to this. The player of this game has solved about half of the puzzle with "mines" on the right hand size correctly identified:



Full details of the text-menu driven implementation and input/output examples can be found in the comments in the start up code.
Here are couple of websites showing how the game is played:
http://en.wikipedia.org/wiki/Minesweeper

For other online demo, have a look here:
http://www.freeminesweeper.org/minecore.html

Consider a text example:
The game shows a number in a square which tells you how many mines there are adjacent to that square. Each square has at most eight adjacent squares. Figure below shows that the **4 x 4** field on the left contains two mines, each represented by a "*****" character. If we represent the same field by the number of adjacent squares described above, we end up with the field on the right

```
*...        *100
....        2210
.*..        1*10
....        1110
```

Note that safe squares are denoted by "`.`", and mine squares by "`*`".

You need to write a complete program that first reads in two integer inputs ***m*** and ***n*** ***(0<n; m≤15)*** that define the number of rows and columns of the field respectively. Each of the next **n** lines contains exactly **m** characters, representing the field. The function then reads in another integer input ***numMines*** that represents the number of mines in the above field. The mine squares are randomly placed on the field. The output should be a field of the same size, with the "`.`" characters replaced by the number of mines adjacent to that square.

Example:

```
Mine sweeper
------------
The number of lines (m): 4
The number of columns (n): 4
The number of mines (numMines): 2

*...      *100
....      2210
.*..      1*10
....      1110

Try again? (y/n):
```

Requirement #3 – Implementation of Minesweeper start up code - 75 marks

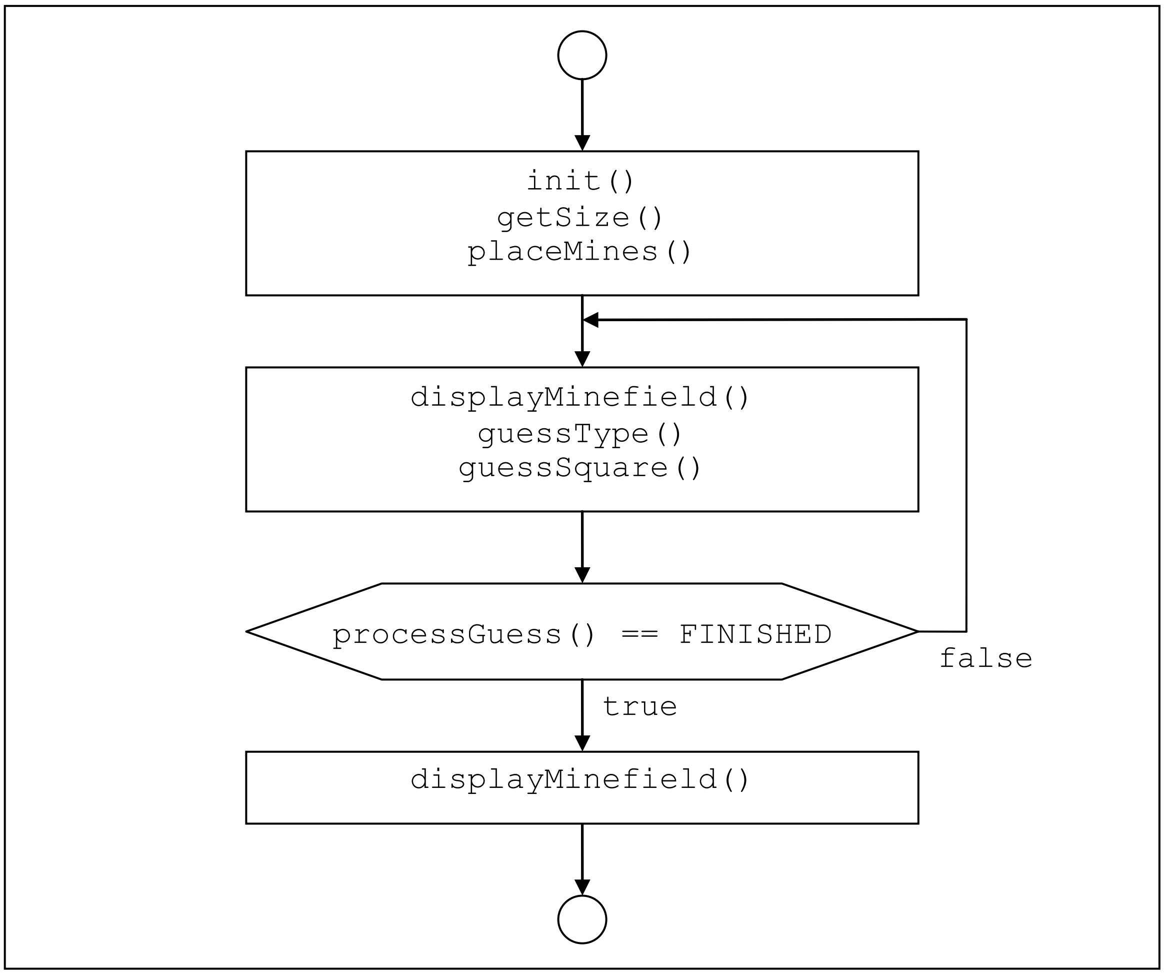
The start up code for the Minesweeper game has empty function definitions for the following seven functions. It is your job to implement them according to the descriptions in the start up code.

- `init()`
- `getSize()`
- `placeMines()`
- `displayMinefield()`
- `guessType()`
- `guessSquare()`
- `processGuess()`

You can get full marks for each of these only if your program runs them correctly. These functions are worth 10 marks each with the exception of the init() function (5 mark) and the processGuess() function (20 marks).

Requirement #4 – Program flow and execution - 10 marks

To gain full marks for this requirement, you will need to fully implement your main() function so that it runs your Minesweeper game according to the flow chart below.



Requirement #5 – Functional Abstraction -5 Marks

We encourage the use of functional abstraction throughout your code for both programs. It is considered to be a good practice when developing software with many benefits. Abstracting code into separate functions reduces the possibility of bugs in your project, simplifies programming logic and eases the need for debugging. We are looking for some evidence of functional abstraction throughout your code. As a rule, if you notice that you have the same or similar block of code in two different locations of your source, you can abstract this into a separate function.

Requirements #6-9 – General requirements – 20 marks

You must implement Buffer Handling, Input Validation, and Coding Conventions as per the requirements listed on the "Assignment General Information" sheet (available on Blackboard. These requirements are going to be weighted as:

- Buffer Handling: 6 marks
- Input Validation: 8 marks
- Coding Conventions: 6 marks

Deliverables and Submission Details

Submission date/time:

Submission details for assignment #1 are as follows. Note that late submissions attract a marking deduction of 10% per day for the first 5 university days. After this time, a 100% deduction is applied.

CATEGORY	DUE DATE/TIME	PENALTY
On time	**Sunday 12/7/2015 11:59pm**	**N/A**
1 day late	Monday 13/7/2015 11:59pm	10% of total available marks
2 days late	Tuesday 14/7/2015 11:59pm	20% of total available marks
3 days late	Wednesday 15/7/2015 11:59pm	30% of total available marks
4 days late	Thursday 16/7/2015 11:59pm	40% of total available marks
5 days late	Friday 17/7/2015 11:59pm	50% of total available marks
6 or more days late	Not accepted	100% of total available marks

We recommend that you avoid submitting late where possible because it is difficult to make up the marks lost due to late submissions.

For assignment #1, you need to submit 5 files:

- **hangman.c:** This file will contain your implementation of the hangman program.
- **hangman.h:** This file is your header file for the hangman program.
- **minesweeper.c:** This file will contain your implementation of the minesweeper program.
- **minesweeper.h:** This file is your header file for the minesweeper program.
- **README:** This file contains additional information about your program that must be filled out.

Your assignment must compile and execute cleanly (no warnings) on **the Linux core teaching servers**. using the –ansi, –Wall., –pedantic flags of the gcc compiler.

(Note that “Linux servers testing” is recommended a few weeks before submission date. Note that you may also use the **–lm** flag if required).

Compile:
```
% gcc -lm -ansi -Wall -pedantic hangman.c -o hangman
```

Execute:
```
%./hangman
```
Compile:
```
% gcc -lm -ansi -Wall -pedantic minesweeper.c -o minesweeper
```

Execute:
```
%./minesweeper
```

You are expected to have a partly modularized solution using all functions and definitions in the start up code provided as a minimum.

Permission to change the start-up code is not normally given, unless there is very good reason (i.e: a bug). You will need to build upon the code that is provided, instead. If you have any concerns about the start up code, please post to the Blackboard discussion board.

Assignment Regulations

- You may refer to textbooks, notes, work in study groups etc. to discover approaches to problems, however the assignment should be your own individual work.
- Where you do make use of other references, please cite them in your work. Students are reminded to refer and adhere to plagiarism policies and guidelines: RMIT CS&IT Academic Integrity: http://www.cs.rmit.edu.au/students/integrity/

How to Submit

This assessment task is due at 11.59pm on Sunday 12th July (the end of week 6).

Late submissions must first be approved by the Online Programs Administrator by completing the “Assignment Extension Form” at this URL:

http://oua.cs.rmit.edu.au/procedures/forms.html

Assignments may be accepted up to 6 days after the deadline has passed; a late penalty of 10% will apply for each day late. Submissions over 4 days late will not be marked.

Emailed submissions will not be accepted without exceptional circumstances.

Submission will take place through Weblearn. It should be a compressed archive called - assignment1.zip- containing hangman.c, hangman.h , minesweeper.c, minesweeper.h and the README file. Please submit in one of the following formats:

- ZIP compressed file (.zip; created by Windows XP, WinZIP Legacy, MacOS X, etc.)
- tar/gzip compressed file (.tar.gz; created by the Unix tar and gzip tools)

***Important Note:* other compression formats, such as (but not limited to) RAR/WinRAR (.rar) and proprietary WinZip (.zipx) will result in mark deductions.**

Submit a README text file as described above. The README should inform the marker of the development environment used for completing the assignment,. Remember the marker will be using gcc with the flags -ansi -pedantic -Wall

Also, each README should state any problems you encountered that may help the marker.