**Text-based MineFinder**

Programming Assignment 1

CPSC 1020, Fall 2017

Due: Wednesday, September 27, 2017

**Learning Objectives:**

You will gain experience using the following programming concepts:

* Multiple files
* File pointers
* File I/O
* Command line arguments
* Pointer
* 2D arrays
* Dynamically allocating memory
* Functions
* Creating user menu
* Validating user data
* Many other basic programming concepts

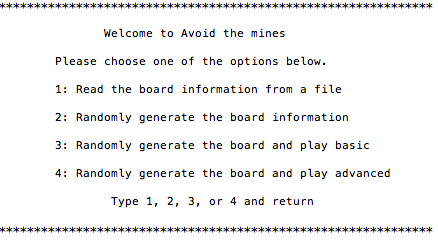
**Program Overview:**

The game of Minesweeper (http://www.freeminesweeper.org/minecore.html) is traditionally a one-player game played against a computer. There are several covered squares and several of these squares have “mines” in them. The goal of the game is to label all the mines without “stepping on” any.

The computer hides the mines in the minefield. The user does not know where the mines are. The user then selects squares in the minefield. If the square contains a mine, the game is over. If the square does not contain a mine, the computer puts a number on the square that indicates how many mines are immediate neighbors to that square (or leaves the square blank if there are no immediate neighbors).

This program is divided in to 4 parts, referred to as Selections. **You are required to complete Selections 1 and 2**. The remaining 2 are extra credit.

You will create a function that provides the user with all 4 selections (described below). If you only implement the basic requirements (the first 2) you should still include the remaining selections on the menu. If the user chooses 3 or 4 and you chose not to implement that selections, you should print a message stating you did not implement this feature. You can title your game whatever you wish. Below is an example of a basic menu. You may be as creative as you wish with the menu or just use this basic format.



Below is a description of each menu selection.

**Selection 1 - Basic**

Read in the board information from a file. The first line of the file will contain two integers to tell you how big the board is (first height (row), then width(col)). The dimensions will not exceed 20 x 20. However, the dimensions could be something like 20 X 10. The remaining lines in the file will contain the minefield, where 0 means there is no mine and 1 means there is a mine. Your goal is to make the computer print out the completely revealed board for that minefield, where X indicates the presence of a mine, and the remaining squares contain a count of how many mines are in its immediately neighboring cells.

**Example:**

You will use command line arguments to provide the file name to be read in. I will provide a sample input file for you to use to test your program. Your output should be printed to a file provided through command line arguments.

**Selection 2 –Randomly generate the board**

Randomly generate the input file rather than reading in the information from a file. You will need to ask the user for the height (rows) and width (col) of the board. The minimal row/col should be 5 and maximum should be 20. The number of mines to be randomly place on this board will be 15% of the size of the board. In other words, height\*width\*.15. Make sure you do not allow to place multiple mines in one place on the board. To complete this selection, you will need to look up the rand and srand functions. The URL’s listed below are examples of how to randomly generate information. <https://www.tutorialspoint.com/c_standard_library/c_function_rand.htm><https://www.codingunit.com/c-reference-stdlib-h-function-rand-generate-a-random-number><http://cforbeginners.com/random_C.html>

After you have generated the file then create the solution and print it to the output file provide on the command line.

**Example:**

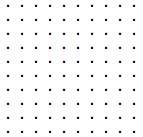
Enter the Height and Width you wish the board to be!

The Height and Width should be between 5 - 20

You must place a space between the height and width.

**Selection 3 - (5 points extra credit) Basic playable**

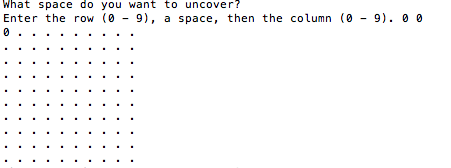
In addition to completing the requirements for Selection 2, you will need to provide code allowing the game to be playable in a basic form. As an example, after the game board has been randomly generated and the solution created, you should generate a playable board using **“.”**, as shown below. The example below is for a 10 by 10 board. While it is not required, feel free to add a row of numbers across the top and left side of the board to denote the rows and columns. This will make it easier for the user to determine the row/col input.



Ask the user to enter the location they want to uncover by entering the row and column.

If the row - column is an “X” then declare that the user has hit a mine and end the game. If the guess is not an “X” then replace the “.” with the value and continue, as shown.

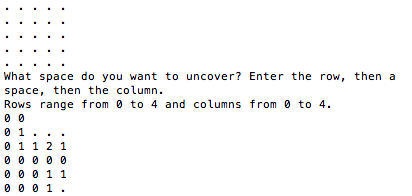
**Example:**



If the user is able to uncover all spaces without hitting a mine, print a congratulatory message.

**Selection 4 - (10 points extra credit)**

In addition to completing Selection 2 you will need to provide an selection making the game playable in the fashion of the original minesweeper game. As an example:



Again, while it is not required, feel free to add a row of numbers across the top and left side of the board to denote the rows and columns. This will make it easier for the user to determine the row/col input.

In the above example, the user chose row 0 and column 0. This selection basically looks for a 0 and any of it’s neighbors that are not a mine. Using recursion, it continues looking for 0’s revealing all non-mine neighboring values.

**Other Programming Requirements**

When completing this program there are a few concepts you must use.

1. You must have a minimal of 4 files.
   1. main.c – this file should contain only the main. The main should have a minimal amount of code. If you have an algorithm (a set of steps necessary to complete a task) that is more than a couple lines of code (excluding loops) you should create a function for this routine.
   2. functions.c – this file contains the implementation of the functions used with this assignment
   3. functions.h – this file contains the prototypes for the functions used in this program. You should use #ifndef and #endif in this file. (see Include Guards explanation below.)
   4. README – this is a plaintext file. This file should be all caps and it does not need an extention. You should use this file to:
      1. Describe any problems you encountered when creating this program.
      2. Describe how you solved the problems you encountered.
      3. This is your opportunity to describe your thoughts concerning this assignment. Did you like the assignment? Why/Why not? If you could change something about this assignment what would that be?
2. You must use file pointers to open the input and output files. These files will be determined using command line arguments.
3. You must use 2 dimensional (2D) arrays.
4. You must dynamically allocate memory for all 2D arrays.
5. When necessary, you must validate user data.
6. As stated above you must use a menu to determine the selection the user wishes to run.
   1. Everyone must complete Selection 1 and 2.
   2. Selections 3 and 4 are extra credit. If you successfully implement Selection 3 (5 points EC) and 4 (10 points EC) you will receive up to 15 extra credit points.

**A Note on Academic Integrity**  
I think this assignment is a really good way for you to practice using pointers, functions, memory allocation and all other computing concepts list above. I am aware that there are plenty of C implementations of Minesweeper available online. However, copying and pasting portions of them won’t help you understand the material. (Less importantly, but still worth considering, I’ll probably figure it out if you do copy others code. Please review the academic integrity policy in the syllabus.)

This is an **INDIVIDUAL** assignment. You may **NOT** seek help on this assignment from anyone other than **myself or a CPSC 1020 lab TA.**

**Formating:**

* You should include a header in all files similar to the following:

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

\*Your name \*

\*username

\*CPSC 1020 Fall 2017 \*

\*Due Date: \*

\*Instructor: Dr. Yvon Feaster \*

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

* Your program should compile with no errors and no warnings:
  + You should compile the program as follows:

**gcc main.c functions.c –o main –Wall**

* + To run the program type ./main *infilename outfilename*
  + Prior to submitting this assignment, you should test it on the SoC servers. I will be grading on the school of computing servers.
* Your program should be well documented using blocks of comments.
* You should use meaningful variable names.
* No line of code should be longer than 80 characters.
* Proper and consistent indention is required.

Failure to follow any of the instructions in the Required Code Formatting section will result in a reduction of points.

**Handin Instructions**

Use the tar utility to tar.gz all source files. **Do not tar an entire directory!**

Name your tarred file: *tar –czvf <username>\_PA1.tar.gz* *list of files to tar* (ex. yfeaste\_PA1.tar.gz)

Use handin (https://handin.cs.clemson.edu) to submit your archive

**Include Guards**

A common problem in programs is that a header file is required in multiple other header files that are later included into a source code file, with the result often being that variables, structs, classes or functions appear to be defined multiple times (once for each time the header file is included). This can result in a lot of compile-time headaches. Fortunately, the preprocessor provides an easy technique for ensuring that any given file is included once and only once.   
  
By using the #ifndef directive, you can include a block of text only if a particular expression is undefined; then, within the header file, you can define the expression. This ensures that the code in the #ifndef is included only the first time the file is loaded.

#ifndef FUNCTIONS\_H

#define FUNCTIONS\_H

/\* code \*/

#endif

(Note that there is an **n** in #ifndef--it stands for "if not defined").

Normal convention is when using #ifndef XXXXXX you will use the header file name you are currently in. You don not necessarily have to use the file name.

With minor changes, this explanation was borrowed from the following: <http://www.cprogramming.com/tutorial/cpreprocessor.html>