Assignment - Dynamic Memory Allocation

Run-time memory allocation and arrays of arrays

Purpose of the exercise

This exercise will help you do the following:

1. Develop skills in allocating memory in run-time for dynamic arrays.
2. Practice use of structures and dynamic 2D arrays.

Overview

Minesweeper is a tile-based computer game originating from 1960s, notable for its long history, widespread adoption across multiple platforms and its implementations bundled with the Microsoft Windows operating system since Windows 3.1 (1992) until Windows Vista (2006), currently still available from Microsoft Studios in Microsoft Store.

Minesweeper is a game where the map consists of a grid of tiles with bombs distributed randomly across the grid. Each tile consists of one of the following:

- If a tile contains a bomb, the symbol `X` indicates the **bomb**.
- If a tile does not contain a bomb, a **number** indicates how many bombs are located in adjacent tiles.

Initially, all tiles are hidden. During a game the player reveals tiles one by one. The player wins the game when they reveal all the tiles without bombs, and loses when a bomb is revealed. In some implementations an empty tile indicates the number `0`; in yours an empty tile will indicate a tile that has not been revealed yet.

Visit http://minesweeperonline.com/ to experience how this game is played.

Below you can see a valid 3x3 map of this game:

Column \ Row	**[0]**	**[1]**	**[2]**
[0]	**X**	**X**	1
[1]	2	3	2
[2]	0	1	**X**

In this assignment, your task is to complete various functions that support test drivers and eventually the main driver of the program that non-interactively simulates the game. In your implementation, the functions will take in the *height* and the *width* of the map, followed by the probability of whether a tile does **not** contain a bomb (the higher the probability, the less likely there is going to be a bomb in each tile; `0` probability means that every tile contains a bomb).

A map is represented as a grid: a dynamic array of rows, where each row is a dynamic array of tiles.

Each tile in a map should be described using the following `struct` data type:

```
typedef struct
{
    char state;
    bool is_visible;
} Tile;
```

The `state` data member represents either an ASCII code of a digit with the number of bombs adjacent to the tile, or `'X'` indicating the bomb on the tile itself. The `is_visible` data member indicates whether the tile has been revealed (`true`) or remains hidden (`false`).

A map is stored as an object with the following data type:

```
typedef Tile* Row;
typedef struct
{
    unsigned short int width;
    unsigned short int height;
    Row* grid;
} Map;
```

Because the `grid` data member is a pointer to a dynamic array where each element is a dynamic array, you can address individual tiles in it as if they are stored in a two-dimensional array.

Task

1. Download the assignment source code files:

   Go to the course page in *Moodle* - DigiPen (Singapore) online learning management system, download a zipped archive and extract its files into a Microsoft Windows directory *c:\sandbox* (adjust the path as suitable for your system). The included files are:

   - *minesweeper.c*
   - *minesweeper-utils.h*
   - *test-1-allocate_and_destroy.c*
   - *test-2-print.c*
   - *test-3-initialize.c*
   - *in-2-0.txt - in-2-4.txt*
   - *expected-output.txt*
   - *expected-output-1.txt - expected-output-3.txt*
2. Using the Microsoft Windows command prompt navigate to the *sandbox* folder. Then type `wsl` to open Linux bash in the same current directory.
3. From Linux open *minesweeper-utils.c* for editing. This is the file you have to complete.
4. Inside *minesweeper-utils.c* implement the functions as declared in *minesweeper-utils.h*:

   - `create_map()`

     The function allocates the map with a dynamic array of `height` pointers to dynamic arrays (`Row` objects) of `width` count of `Tile` objects.

     ```
     Map* create_map(unsigned short int width, unsigned short int height);
     ```

     Before testing this function implement `destroy_map()` and `set_tile()`.

   - `destroy_map()`

     The function deallocates all the memory allocated for the map.

```
void destroy_map(Map* map);
```

Before testing this function implement `create_map()` and `set_tile()`.

- `set_tile()`

  The function sets the initial state of a tile for testing purposes.

```
void set_tile(
    Map* map,
    unsigned short int column,
    unsigned short int row,
    char state,
    bool is_visible);
```

  The data members `state` and `is_visible` of the tile at given `column` and `row` will be set from the function's parameters.

  Before testing this function implement `create_map()` and `destroy_map()`. To test it use the *test-1-allocate_and_destroy.c* test driver:

```
gcc -Wall -Wextra -Werror -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -o main minesweeper-utils.c test-1-allocate_and_destroy.c
./main > actual-output-1.txt
diff expected-output-1.txt actual-output-1.txt --strip-trailing-cr
```

  Your actual output must exactly match the contents of the expected output. Use the *diff* command to compare the files; no differences in the output should be reported.

- `print_map()`

  This function prints the tiles in row by row.

```
void print_map(Map const* map);
```

  If a tile is invisible, a space character is printed instead of the tile state symbol. Note that the function does not check the validity of the map; it simply prints what is given to it.

  To test this function use the *test-2-print.c* test driver:

```
gcc -Wall -Wextra -Werror -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -o main minesweeper-utils.c test-2-print.c
./main > actual-output-2.txt
diff expected-output-2.txt actual-output-2.txt --strip-trailing-cr
```

  Your actual output must exactly match the contents of the expected output. Use the *diff* command to compare the files; no differences in the output should be reported.

- `initialize_map()`

  This function takes in a map that has been allocated and fills it with values for each tile.

```
void initialize_map(Map* map, float probability);
```

  The following pseudocode shows the algorithm you are supposed to implement:

```
for each row i
    for each column j
        set a tile at the position i, j to not visible
        r ← rand()%100
        if r >= probability * 100
            set a tile at the position i, j to a bomb
        else
            set a tile at the position i, j to 0 adjacent bombs
        end if
    end for
end for
for each row i
    for each column j
        if a tile at the position i, j is not a bomb
            b ← the number of bombs in adjacent tiles
            increase the bomb count of the tile by b
        end if
    end for
end for
```

As shown above, use `rand()%100` to get a random number where `rand` is a function to generate a pseudo-random number declared in `stdlib.h`.

To test this function use the *test-3-initialize.c* test driver:

```
gcc -Wall -Wextra -Werror -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -o main minesweeper-utils.c test-3-initialize.c
./main > actual-output-3.txt
diff expected-output-3.txt actual-output-3.txt --strip-trailing-cr
```

- `reveal_all_tiles()`

  The function sets all tiles of a map as visible.

  ```
  void reveal_all_tiles(Map* map);
  ```

- `all_empty_tiles_visible()`

  The function tests whether all non-bomb tiles have been revealed. It returns `true` if and only if all non-bomb tiles are visible (bomb tiles can be visible or not); otherwise, it returns `false`.

  ```
  bool all_empty_tiles_visible(Map const* map);
  ```

- `is_bomb_tile()`

  The function reveals the tile at the specified `column` and `row` (both indexed from 0). If the tile contains a bomb, the function returns `true`; otherwise, it returns `false`.

  ```
  bool is_bomb_tile(
      Map* map,
      unsigned short int column,
      unsigned short int row);
  ```

  In the event that `column` or `row` specified is out of range, the function should print an error message to the output stream:

```
Error: wrong column or row specified.
```

5. Compile the main driver of the program:

```
gcc -Wall -Wextra -Werror -Wconversion -Wstrict-prototypes -pedantic-errors -std=c11 -o main minesweeper-utils.c minesweeper.c
```

6. Run the executable file:

```
./main > actual-output.txt
```

7. Use *diff* to compare the actual output with the expected output::

```
diff expected-output.txt actual-output.txt --strip-trailing-cr
```

   Make sure that the output matches **exactly**.

Submitting the deliverables

You have to upload a complete file *minesweeper-utils.c* to *Moodle* - DigiPen (Singapore) online learning management system, where the file will be automatically evaluated.