REPORT FOR ASSIGNMENT 3
CSE5CES

VU PHUC THANH TRAN
ID: 18869804

May 19, 2017

Question 1

How function get_move() was implemented?

Function get_move() has 5 arguments which are:

+ A pointer which will point to variable (coordinate x) in function main().
+ A pointer which will point to variable (coordinate y) in function main().
+ A pointer which will point to variable (flag_status) in function main().
+ Max value of coordinate x (value height in struct Game)
+ Max value of coordinate y (value of width in struct Game)

- Inside function get_move(), there is a do...while loop for receiving and validating input from users.
- In this do...while loop, a statement will be displayed for asking users to enter coordinates x (height), y (width) and flag (0/1) by using function scanf().
- After using scanf(), a code statement is used for cleaning keyboard buffer. That will make sure that other functions scanf() will not operate in wrong way.
- After receiving those inputs from users, they will be validated to make sure that there is any bad effects on the others part of the program.

- A variable named ‘check‘ are created to get return result from function scanf()
    + IF check is equal to zero, THEN that means scan() can not get any value from keyboard buffer and storing to the variable.
- There are three conditions for do...while loop. If one or more than one of them are fail, then the program will start to ask users about inputs again. If all of them are successful, then the program will get out of loop and return values for three variables in function main(): x, y, flag_status via pointers.

Condition 1: IF check is equal to zero
OR
Condition 2: IF x (input) or y (input) are larger than max value of x and y
OR
Condition 3: IF flag (input) is not equal to zero and flag (input) is not equal to one

Why get_move() was implemented that way?

- Do...while loop was used because it does code block inside first then check conditions at the end of the loop. Therefore, all code block for asking users, receiving inputs and validating could be placed inside the loop and will be displayed again if users enter invalid input.
- After using scanf(), there need to be a code statement for cleaning buffer. Because, sometimes, scanf() just takes part of keyboard buffer or does not take any thing and leaving all data in buffer. Therefore, cleaning buffer will help to avoid unpredicted effects on the program.
- 3 of 5 arguments (get_move()) are pointers which point to those variables in function main(). This assignment requires to get and return three variables to function main(), so,

pointer need to be used.

Question 2

How step 6 and 7 in function main were implemented?

- First of all, to create a two dimensions array for storing mine_board, there need to be a pointer with two stars (**) (pointer which points to another pointer). In the program, a pointer with 2 stars was declared.

```
struct Location** mine_board;
```

- First pointer will point to an array which is assigned by function malloc() with data type is struct Location**. The length of this array is equal to height of the game board + 2 (input from users).

```
mine_board = (struct Location**)malloc((game.height + 2) *
sizeof(struct Location));
```

- Each element in that array is also a pointer, that pointer will pointer to an array assigned by function malloc() with data type is struct Location*. Then, a loop can be used for assigning each element in first array to an array (by function malloc()). The length of each array is equal to width of game board + 2 (input from users).

```
for(i = 0; i < (game.height + 2); i++)
        {
                mine_board[i] = (struct Location*)malloc((game.width + 2) *
                sizeof(struct Location));
                if(NULL == mine_board[i])
                {
                        printf("Error of malloc() function\n");
                        return 1;
                }
        }
```

- After each time using function malloc(), an if...else will be used for comparing the variable which was assigned by malloc() with NULL value. If that variable is equal to NULL, the program will exit and displaying an error statement.
- Finally, a two dimension array was created.

Why are step 6 and 7 implemented that way?

- Game field need to be created after receiving height and width from user's need. Hence, malloc() will be used to create array which is based on user's need.
- First, an array which presents for height of game board will be created by malloc().
- Then for each row, malloc() need to be operated again to create array (width)(column) for

each row. This part will be put on a loop.
- When using malloc() for creating arrays, two more rows and two more columns were added. Thus, these rows and column will be used for borders around game field.

How function make_move() run?

There are 4 arguments in function make_move():

1 - a variable with data type struct Game
2 - a pointers with two starts which will points to variable mine_board in function main()
3 - value of coordinate x
4 - value of coordinate y

- When make_move() is called, value of game (struct Game) in function main will be passed to first argument. Then, value of coordinate x and value of coordinate y in function main will be passed to third and fourth arguments as well. The pointer (second argument) will point to variable mine_board in function main().
- Function make_move() will take coordinate x, y and setting REVEALED bit to 1 at this position.
- Checking if there is a mine of this location or not
+ Yes: set statusOfPlayer to zero (gameover)
+ No
. Checking if there is a flag, then remove that flag.
. Checking if value of this location is equal to zero, then call function uncover_zeros().