CSE102 – Computer Programming

Homework #9

Dynamic Arrays

Due Date: 05/06/2023

Hand in: A student with number 20220000001 should hand in a zip file named 20220000001.zip for this homework.

Homework Description: In this assignment, you are required to develop the snake game using C programming language. The game area is represented by a 10 x 10 x m 3D board, and the snake moves within this area. The user controls the snake using the 'w' (up), 'a' (left), 's' (down), and 'd' (right) keys, aiming to achieve the longest snake possible. The snake grows longer by consuming baits placed on the map. However, there are obstacles on certain blocks that can end the game. The game ends under three conditions: (I) When the snake's head touches its body, (II) when the snake collides with a wall, or (III) when the snake encounters an obstacle longer than itself.

For this assignment, you are only allowed to use dynamic arrays. Failure to adhere to this requirement will result in a score of 0, even if all other requirements are met. The details of the assignment are provided in the following sections.

Important! You must use the given function names to implement the different parts of the assignment. It is not permitted to use any additional helper or wrapper functions apart from the ones provided.

You are required to define a point structure. The snake is implemented as a dynamic array of point structures. The point data structure consists of two integer variables: row and col. These variables are used to store the position of a block occupied by a part of the snake. The array contains *n* points, where *n* is the current length of the snake.

The board consist of blocks. The block structure has two variables: type (char) and value (int). The type can have one of the three values: 'o' for obstacle block type, 'b' for bait block type, and 'e' for empty block type. The value represents the height of the obstacle, and it is zero for empty and bait blocks.

Part 1. [30 pts] Initialize The Board

This part should be implemented in the **init_board()** function. This function generates and returns the game board, which is a 3D dynamic array of type **Block**. The board consists of 10 rows and 10 columns, with the possibility of having nested blocks in each cell. Therefore, the cells should be implemented using dynamic arrays of **Block**. The board needs to be initialized in such a way that each cell contains an empty block. Finally, randomly change the type of two blocks on the board to create one obstacle and one bait. Avoid placing them on the top-left block where the snake will start.

Part 2. [20 pts] Draw The Board

This part should be implemented in the **draw_board()** function. The function takes the game board as input and prints it in the specified format below. The function uses "-" for horizontal lines and "|" for vertical lines to draw the edges of the game board. When drawing the cells:

- If the cell is of type "empty", nothing is printed.
- If the cell is captured by the snake's body, "X" is printed.
- If the cell is captured by the snake's head, "O" is printed.
- If the cell is of type "bait", "." is printed.
- If there is an obstacle in the cell, the number of nested blocks is printed.

Part 3. [40 pts] Game Play

This part needs to be implemented in four functions: **play(), move(), check_status(),** and **update()**.

play(): This is the main function for game play. It takes the initialized board as input and does not return anything. The function initializes the snake with a length of 1 block at the top-left of the board. It asks the user for a new move until the game is over and draws the board using the draw_board() function before each move. After each move, the current state of the game is checked using the check_status() function. If the game is not over in the current state, then the board and snake are updated in the update() function.

move(): This function takes the snake as input and applies the next move obtained from the user. The next move should be obtained from the user within this function, and the head of the snake should be set to its next position based on the given direction. The snake can turn in one of the four directions except the direction that goes through its own body. If the user tries to make such a move, it should be ignored.

check_status(): After the move is made, the consequences of the last move need to be checked. This function takes the snake and the board as input and returns 1 if the game is over, otherwise it returns 0. In this function, three conditions need to be checked:

1. Does the head of the snake hit its own body?
2. Does the snake collide with the wall?
3. Does the snake encounter an obstacle that has x number of blocks, where x is greater than the length of the snake?

The game is over if any of these conditions are met, so the function returns 1 in that case.

update(): If the game is not over, the play() function calls the update() function. In this function, the program updates the coordinates of the snake's body, considering whether the snake ate a bait in the last move or not. Then, the function updates the bait and obstacles on the board. If the snake ate a bait in the last move, the length of the snake is increased by one block. Otherwise, the rest of the body is shifted to keep the body and head connected. There can be only one bait at a time on the board. This function also updates the obstacles every 5 moves. The obstacles appear randomly on the board, and there can be a maximum of 3 obstacles at a time. Additionally, the height of obstacles can be increased by adding new obstacle blocks on top of existing ones. If the snake encounters an obstacle that is shorter than its body, it can capture the block by destroying the obstacle. This is done by deleting the nested blocks on the obstacle, setting the block type to empty, and the value to 0. Note that neither obstacles nor bait can appear in blocks that are currently occupied by the snake.

Part 4. [10 pts] Main function

You have to write a **main** function that calls the **init_board()** function to initialize the board and retrieves the board. Then, the board is passed to the **play()** function to start the game. You don't need to create a menu for this homework; the game should start automatically when the program runs.

Notes:

**** You are required to create a demo video showcasing all the necessary functions in the game, such as nested obstacles, eating bait, confronting obstacles, hitting walls, and encountering the snake's own body. The video should not exceed 5 minutes in length. Please upload the videos to YouTube with unlisted visibility and share the link by including it in a text file within the homework document.**

****Do not forget to prepare a makefile (-50 points)**

General Rules:

1. Make sure to include appropriate comments and variable names in your code to make it easy to understand.
2. The program must be developed on given version of OS and must be compiled with GCC compiler, any problem which rises due to using another OS or compiler won't be tolerated.
3. Note that if any part of your program is not working as expected, then you can get zero from the related part, even it is working partially.
4. Zip your homework files before uploading them to MS Teams. The zip file must contain the C file with your solution and screenshots of the valid outputs of the program.
5. You can ask any question about the homework by sending an email to b.koca@gtu.edu.tr or by using the homework channel on MS Teams page of the course.