BLG 233E DATA STRUCTURES AND LABORATORY

EXPERIMENT 9 – RECURSION

IMPORTANT REMINDERS

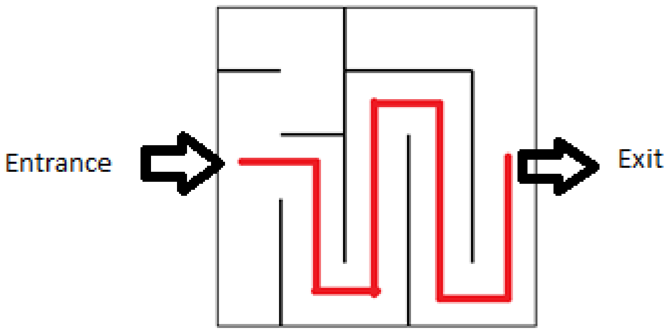
1. It is not allowed to use USB sticks during the lab sessions.
2. You should unplug your ethernet cables during the lab sessions.
3. Any reference book or help material (C++) is allowed.
4. In this lab session, you will work in pairs.

In this experiment, you are required to implement a recursive maze generation algorithm and an algorithm that finds the path in this maze through the exit. Different from the example that you covered in the lecture, we will use a totally new maze struct that is given below.

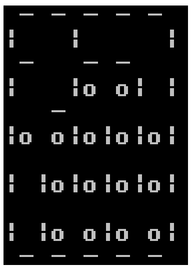
```
typedef struct cell
{
	bool left, right, up, down;   // right = false when a wall does not exist otherwise, right = true means there is a wall
	char val;          // initially this is empty (' '), when the cell is used to reach the exit, this becomes 'o'
	bool visited;     // this variable will be used while generating maze randomly (look the pseudo code for further information)
};

cell m[MAZESIZE][MAZESIZE]; // this is the maze
```

Print function for printing this maze to the screen is also given (look at the end of the sheet). For example, for the maze in the figure (a), output of the print function would print as in (b). In figure (a), red line is the path through the exit so in figure (b), paths that corresponds to this red line is marked with a ‘o’. Here, m[2][0] is the initial point and m[2][4] is the exit.



(a)



```
 _ _ _ _ _
|   |     |
 _   _ _
|   |o o| |
   _
|o o|o|o|o|

| |o|o|o|o|

| |o o|o o|
 _ _ _ _ _
```

(b)

Initially, all the walls in the cells exist (left, right, up and down variables are true). In the generation phase, your algorithm should remove some of the walls in order to create a maze. A pseudo code is given for generating a random maze below.

```
Make the initial cell the current cell and mark it as visited
While there are unvisited cells in the maze
    If the current cell has any neighbours which have not been visited
        Choose randomly one of the unvisited neighbours
        Push the current cell to the stack
        Remove the wall between the current cell and the chosen cell (remind that you should modify the related variables in both neighbour cells)
        Make the chosen cell the current cell and mark it as visited
    Else if stack is not empty
        Pop a cell from the stack
        Make it the current cell
    Else
        Pick a random cell, make it the current cell and mark it as visited
```

In the given algorithm, you need to use a stack for generating the maze. Therefore, you should also implement your stack struct. Since the stack's maximum size is MAZESIZE*MAZESIZE, you are allowed to use either linked list or array representation.

After generating the maze, you should print the maze to the screen and ask the user about the entrance and exit locations in the maze. Then your recursive function that finds a path in the maze from a start position through the exit finds a path and prints the maze with the path to the screen.

```cpp
void print(cell m[][MAZESIZE])
{
	for(int i = 0; i < MAZESIZE; i++)
	{
		for(int j = 0; j < MAZESIZE; j++)
		{
			if(m[i][j].up)
				cout << " -";
			else cout << "  ";
		}
		cout << endl;
		cout << "|";
		for(int j = 0; j < MAZESIZE; j++)
		{
			if(m[i][j].right)
				cout << m[i][j].val << "|";
			else
				cout << m[i][j].val << " ";
		}
		cout << endl;
```

```
        }
        for(int i = 0; i < MAZESIZE; i++)
        {
                if(m[MAZESIZE-1][i].down)
                                cout << " -";
        }
        cout << endl;
}
```