**APS 105 — Computer Fundamentals**
Lab #6: 2D Arrays
Winter 2023

You must use `examify.ca` to electronically submit your solution by 11:59 pm on **Saturday, March 11, 2023**.

---

## Objective

In this lab, you will be writing one C program. You will gradually develop the program by writing a few functions in each part. In each part, you will only be marked on the functions you submit in that part, not the entire program. Input and output must be done using `scanf` and `printf`.

In the sample output examples that follow, the text that would be entered by the program user is displayed using a **bold** font. The text <**enter**> stands for the user pressing the `enter` key on the keyboard. On some keyboards, the `enter` key may be labeled `return`. When you execute your programs, the user's text will not be displayed in bold and <**enter**> will not be shown.

Once again, place your solution for each part in a separate directory, so that Visual Studio Code will not attempt to compile multiple files and link them together. Each of your directories should have one `.c` file corresponding to your solution for the individual part of the exercise. Please ensure you are selecting the folder with the `.c` you want to compile.

## Game explanation

You are asked to develop a program for a one-player game. The game starts with a 2D board that is filled with `0`s and `1`s. Depending on the board dimensions, the board will be generated. The board contents are hidden from the player. It is required that the user finds a line of `1`s that is equal or greater than the difficulty level. The line can be horizontal, vertical or diagonal. The user will be asked to enter the coordinates until the line of `1`s is found or until the user attempts all boxes, and not find the line of `1`s.

**Points.** The user should find the line of boxes in the fewest possible trials. The points gained is calculated as the number of unattempted boxes in the board. The game ends when the user finds the line of `1`s that is equal or greater than the difficulty level or until the user attempts all boxes, and not find the line of `1`s. In the latter case, the game is invalid. The points will be displayed at the end of the game, if the game is valid.

## Part 1 — Get Input and Print Board

In this part we will develop four functions with prototypes:

```
void printBoard(int Size, int grid[][Size]);
void getLevelAndDimensions(int* Size, int* levelOfDiff, const int maxDim)
bool validRowCol(int row, int col, int Size);
void getInput(int* row, int* col, int Size);
```

The function `printBoard` will print the contents of the board in `grid`. The grid is a square grid, hence the number of rows and columns is the same as the value of `int Size`. Each row should be printed on a different line.

The function `getLevelAndDimensions` will prompt the user to enter the difficulty level of the game, and place it in the variable pointed to by `levelOfDiff`. It should be a number between 1 and `maxDim`. If it is valid, then the function will prompt the user to enter the dimensions of the board. Otherwise, keep prompting the user to enter the difficult level until it is valid. The dimension should be placed in the variable pointed to by `Size`. The dimension should be equal or larger than the difficulty level and less than `maxDim`. If the dimension is valid, then return. Otherwise, keep prompting the user to enter the dimension until it is valid.

The function `validRowCol` returns `true` if the row number in int-variable `row` and the column number in int-variable `col` are valid, and `false` otherwise. The row and column are valid if they are between 0 and `Size - 1`.

The function `getInput` will prompt the user to enter the coordinates of the line the user picks. The coordinates should be placed in the variables pointed to by `row` and `col`. The coordinates should be between 0 and `Size - 1`. If the coordinates are valid, then return. Otherwise, keep prompting the user to enter the coordinates until they are valid. You will need `validRowCol` function in `getInput` function.

**Testing and submission.** To test your functions in VS Code, we have provided a skeleton for your program on Quercus and Examify in `lab6part1-skeleton.c`. However, do not submit the `main` function, function prototypes and includes. Only submit the `printBoard`, `getLevelAndDimensions`, and `validRowCol` and `getInput` functions you developed. We will append the `main` function, function prototypes and includes from our end.

**Example outputs if you test your code using the skeleton given in** `lab6part1-skeleton.c`:
**Example 1:**

```
Enter the difficulty level: 5<enter>
Enter the dimensions of the grid: 5<enter>
11100
10110
01011
10111
10000
Enter user input: 3 5<enter>
Please enter your row and col to be between 0 and Size - 1: 3 2<enter>
row, col = 3, 2
11100
10110
01011
10911
10000
```

**Example 2:**

```
Enter the difficulty level: 24<enter>
Please enter a difficulty level between 1 and 23: 4<enter>
Enter the dimensions of the grid: 2<enter>
```

```
Please enter dimensions >= 4: 4<enter>
1110
1011
0101
1011
Enter user input: 5 7<enter>
Please enter your row and col to be between 0 and Size - 1: 0 3<enter>
row, col = 0, 3
1119
1011
0101
1011
```

**Example 3:**

```
Enter the difficulty level: 1<enter>
Enter the dimensions of the grid: 5<enter>
11100
10110
01011
10111
10000
Enter user input: 3 3<enter>
row, col = 3, 3
11100
10110
01011
10191
10000
```

## Part 2 — Check Score in Direction

In this part, we will develop two functions to find the number of 1s in a given direction. The first function will be used to update the row and col as per the direction we are moving in and if we are moving forward or backward. The second function will be used to calculate the score in a given direction.

```
void updateRowCol(int* row, int* col, int dir, bool forward);
int calculateScoreInDir(int row, int col, int direction, int Size, char userArray[][Size]);
```

The function updateRowCol takes in a pointer to the row and col values, and advance them by one step depending on the direction, which is 0, 1, 2 or 3, and the forward value, which is true or false. The row and col values are updated in place. The function does not return anything. If forward is true and the direction is:

- 0: row is decremented by 1 and col is not changed.
- 1: row is decremented by 1 and col is incremented by 1.
- 2: row is not changed and col is incremented by 1.
- 3: row is decremented and col is decremented by 1.

If forward is |verb|false|, you reverse the incrementing to decrementing and vice versa. For example, if the direction is 1: row is incremented by 1 and col is decremented by 1.
**Note:** In lab 7, we will do something similar but in 8 directions instead of 4 with a forward direction.

The function calculateScoreInDir returns the count of consecutive '1's in a line in a direction starting and including the current row and col. The function takes in the starting points of row and col values, the direction, the Size of the grid, and the userArray, which is the grid having char values either: '0' or '1'. The function returns the number of consecutive '1's in the forward and backward direction. For example, if the direction is 0, the function will check if the current row, col has a value of '1', then count how many consecutive '1's in the vertical line above and below the current row, and return the count of these consecutive '1's. The function does not modify the row and col values.

To test your functions in VS Code, we have provided a skeleton for your program on Quercus and Examify in lab6part2-skeleton.c. However, do not submit the main function, function prototypes and includes on Examify.
**Example outputs:**
**Example 1:**

```
Enter size: 3<enter>
Row: 0, Col: 0, in direction
0 Score: 2.
1 Score: 1.
2 Score: 3.
3 Score: 1.
Row: 0, Col: 1, in direction
0 Score: 1.
1 Score: 2.
2 Score: 3.
3 Score: 2.
Row: 0, Col: 2, in direction
```

```
0 Score: 2.
1 Score: 1.
2 Score: 3.
3 Score: 1.
Row: 1, Col: 0, in direction
0 Score: 2.
1 Score: 2.
2 Score: 1.
3 Score: 2.
Row: 1, Col: 1, in direction
0 Score: 0.
1 Score: 0.
2 Score: 0.
3 Score: 0.
Row: 1, Col: 2, in direction
0 Score: 2.
1 Score: 2.
2 Score: 1.
3 Score: 2.
Row: 2, Col: 0, in direction
0 Score: 0.
1 Score: 0.
2 Score: 0.
3 Score: 0.
Row: 2, Col: 1, in direction
0 Score: 1.
1 Score: 2.
2 Score: 1.
3 Score: 2.
Row: 2, Col: 2, in direction
0 Score: 0.
1 Score: 0.
2 Score: 0.
3 Score: 0.
```

**Example 2:**

```
Enter size: 2<enter>
Row: 0, Col: 0, in direction
0 Score: 2.
1 Score: 1.
2 Score: 2.
3 Score: 1.
Row: 0, Col: 1, in direction
0 Score: 1.
1 Score: 2.
2 Score: 2.
3 Score: 1.
Row: 1, Col: 0, in direction
0 Score: 2.
```

```
1 Score: 2.
2 Score: 1.
3 Score: 1.
Row: 1, Col: 1, in direction
0 Score: 0.
1 Score: 0.
2 Score: 0.
3 Score: 0.
```

**Example 3:**

```
Enter size: 1<enter>
Row: 0, Col: 0, in direction
0 Score: 1.
1 Score: 1.
2 Score: 1.
3 Score: 1.
```

## Part 3 — Check the Game Status

In this part, we develop the function that checks the game status. The skeleton that you can use to test your code on VS Code is on Quercus in `lab6part3-skeleton.c` and on Examify. You are only required to submit the function on Examify. We will append the `main` function, function prototypes and includes from our end. The function prototype is:

```
bool gameStatus(int Size, char userArray[][Size], int difficultyLevel);
```

You will need your implementations of `printBoard`, `validRowCol`, `updateRowCol`, `calculateScoreInDir` to test your gameStatus. However, on Examify, we will only require that you submit `gameStatus`, as we will only test you on that function.

The function takes as input:

- `Size`: size of the array.
- `userArray`: the current status of the board. The hidden places are shown as `'-'`, and the ones picked before by the user are shown as either `'1'` or `'0'`.
- `difficultyLevel`: the difficulty level of the game.

The function should use `calculateScoreInDir` to check for each coordinate/place, if it sees a line of `'1'`s that is equal to difficulty level. If it does, it should return `true`, otherwise it should return `false`.

To test your function in VS Code, we have provided a skeleton for your program on Quercus and Examify in `lab6part3-skeleton.c`. However, do not submit the `main` function, function prototypes and includes on Examify. **Note:** `printBoard` takes in a 2D array of `char` not `int` as in part 1.

If you use the skeleton, for some inputs, *e.g.* size of `4` and difficulty level of `4`, you will find that there will never be a line of `'1'`s that terminates the game. In this case the skeleton will print `"Not a valid game!"` to the user. We will solve this issue in Part 4.
**Example outputs. Example 1:**

```
    Enter size and difficulty level: 3 3<enter>
    Enter row and col: 0 0<enter>
    1--
    ---
    ---
    Enter row and col: 0 1<enter>
    11-
    ---
    ---
Enter row and col: 0 2<enter>
    111
    ---
    ---
    Game over!
```

**Example 2:**

```
Enter size and difficulty level: 6 4<enter>
Enter row and col: 0 1<enter>
-1----
------
------
------
------
------
Enter row and col: 1 1<enter>
-1----
-0----
------
------
------
------
Enter row and col: 1 2<enter>
-1----
-01---
------
------
------
------
Enter row and col: 1 3<enter>
-1----
-011--
------
------
------
------
Enter row and col: 2 3<enter>
-1----
-011--
---1--
------
------
------
Enter row and col: 3 4<enter>
-1----
-011--
---1--
----1-
------
------
Game over!
```

## Part 4 — Create a valid game

In this part you are asked to edit the `main` function in the skeleton we gave you, such that the grid created is a valid game. You are required to randomly select a row and col between 0 and `Size` using the given `getRand` function in `lab6part4-skeleton.c`, and change it to `'1'` until you get valid game given the dimensions and difficulty level chosen by the player. You may write a function for this if you want. Please submit only the `main` function and any extra functions you created for this part.

**Hint:** Use `gameStatus` function to know if you have a valid game.

**Example output is given based on the output of one of the instructor's computers, but it is different from Examify. You have to make your output the same as Examify before submission. Example 1:**

```
Enter the difficulty level: 4<enter>
Enter the dimensions of the grid: 4<enter>

Enter user input: 3 3<enter>
----
----
----
---1
Enter user input: 2 3<enter>
----
----
---1
---1
Enter user input: 1 3<enter>
----
---1
---1
---1
Enter user input: 0 3<enter>
---0
---1
---1
---1
Enter user input: 3 2<enter>
---0
---1
---1
--11
Enter user input: 3 1<enter>
---0
---1
---1
-111
Enter user input: 3 0<enter>
---0
---1
---1
```

```
1111
Game over!
Your score is 9
```

## Marking

This lab will be marked out of 10, 2 marks for Part 1, 3 marks for Part 2, 4 marks for Part 3, and 1 mark for Part 4. The automarker may use multiple test cases in each part, and if this is the case, the marks for this part will be evenly split between the test cases. The test cases used for marking may or may not be the same as the test cases that are made available to you. The deadline is strictly enforced **(11:59 pm on Saturday, March 11, 2023)**, so avoid last minute submissions.