1. IMPORTANT REGULATIONS

- You can not use any library other than ***<stdio.h>***
- ***DO NOT use keywords static.***
- ***You are NOT allowed to use global variables anywhere.***
- A copy of the header file that contains the function prototypes is also provided as ***"lab2_copy.h"***
    - This file is not used in the evaluation.
    - Any change in this file will have no effect in the run, debug or evaluate commands.
- You can use *the* ***"run.c"*** file for testing your functions with the run and debug commands, modify it however you like. This file is not used in the evaluation.
- Please submit your implementations in ***"lab2.c"*** file. **DO NOT touch** the following lines and start coding after these lines:
- `#include <stdio.h>`
- `#include "lab2.h"`
- ```
  /* TASK SOLUTIONS */
  ```

- You can define your own helper functions. However, you **SHOULD NOT** put new function signatures to ***"lab2_copy.h"***, since it is a copied version of ***"lab2.h"*** and changes do not affect the original header file. **Handle everything in the *"lab2.c"* file.**
- Make sure that functions given to you as tasks can be called **"as is". DO NOT change their signature.**
- Make sure that your solution works with the examples that are given to you in the description first.

2. TASKS

In this lab, you will be handling some tasks about character arrays. You will define the behaviour of each task within lab2.c file. The file is already filled with empty task functions.
Each character array ends with an exclamation mark which indicates the end of the sentence. By definition size of the array might (and probably) be larger than the actual length of the sentence.
You may assume the exclamation mark will always be located at the end of the sentence.

Print the Sentence

Method Name: **print_sentence(char * sentence)**

This function prints the content of the given character array. It prints a new line character at the end of the output. Do not print the exclamation mark.
  ```

```
int main(){
    char sentence[256] = "Preliminary Question for the Second Laboratory Exam!";
    print_sentence(sentence);
}
```

Output:

Preliminary Question for the Second Laboratory Exam

Lower the Sentence

Method Name: **to_lower(char * sentence)**

This function lowers the content of the given character array.

```
int main(){
    char sentence[256] = "Preliminary Question for the Second Laboratory Exam!";
    to_lower(sentence);
    print_sentence(sentence);
}
```

Output:

preliminary question for the second laboratory exam

Find all substrings

Method Name: **substrings(char * sentence, char * substr, int * arr)**

Find all occurrences of a substring (another character array) in the given character array. You should use the array (given in the 3rd parameter) which holds starting position of each found substring. The function should work case sensitive.

```
int main(){

    char sentence[256] = "This is the lab exam of CENG140 of Ceng department (in english version)!";

    char substr[10] = "eng!";

    int arr[10] = {0};

    to_lower(sentence);

    substrings(sentence, substr, arr);

    // Content of arr is {25,36,55}

}
```

Remove X Characters

Method Name: **removeX(char * sentence, int * arr, int count)**

This method takes 3 parameters. Original character array, array of remove positions and X number of characters to be removed. From each remove positions remove next X characters from the original character array.

```
int main(){

    char sentence[256] = "This is the lab exam of CENG140 of Ceng department (in english version)!";

    char substr[10] = "eng!";

    int arr[10] = {0};

    to_lower(sentence);

    substrings(sentence, substr, arr);

    // Content of arr is {25,36,55}

    removeX(sentence, arr, 3);

    print_sentence(sentence);

}
```

Output:

this is the lab exam of c140 of c department (in lish version)

Add Some Characters

Method Name: **addSome(char * sentence, int * arr, char * substr)**

Similar to removeX function this function takes also 3 parameters. However, now instead of removing X characters this method adds given substring to each given starting locations.

```
int main(){

    char sentence[256] = "This is the lab exam of CENG140 of Ceng department (in english version)!";

    char substr[10] = "eng!";

    int arr[10] = {0};

    to_lower(sentence);

    substrings(sentence, substr, arr);

    // Content of arr is {25,36,55}

    addSome(sentence, arr, substr);

    print_sentence(sentence);

}
```

Output:

this is the lab exam of cengeng140 of cengeng department (in engenglish version)

Solving BattleShip

Method Name: **battle_ship(char map[mapWidth][mapHeight], int prev_moves[prevCount][2], int next_moves[nextCount][2])**

For this task, you will solve a battleship game. You are aware of the enemy side and you know your previous moves. Now you need to figure the minimum number of attacks (and their locations) in order to win the game.

This function takes 3 parameters map, prev_moves, next_moves which are explained as follows:

- **map:** The enemy side of the map stored in a 2D character array. Empty spaces indicated with a '.' and an enemy ship is given as 'X'.
- **prev_moves:** Your previous moves (attacks) in the game. Which is another 2D array (Moves X 2) each element of the array is a location from the map. Always ends with {-1,-1}.
- **next_moves:** The moves you need to perform in order to win the game. Initially empty, you should be able to fill this 2D array. You **should finally add** {-1,-1} as an indication of the end of the array.

Example:

. . X X X

X X . . .

. . . . .

X X X . X

. . . . X

prev_moves = {{0,2}, {0,3}, {0,4}, {1,1}, {4,4}, {2,2}, {2,3}, {3,1}, {3,2}, {2,4}, {-1,-1}};

Which will change the map as follows (you don't need to change the map this is just an explanation):

. . H H H

X H . . .

. . H H .

X H H . X

. . . . H

Now you should be able to store the remaining ship positions in the next_moves array.

```
next_moves = {{1,0}, {3,0}, {3,4}}


int main(){

        char map[mapHeight][mapWidth] = {{'.','.','X','X','X'}, {'X','X','.','.','.'}, {'.', '.', '.', '.', '.'}, {'X', 'X', 'X', '.', 'X'}, {'.', '.', '.', '.', 'X'}};

            int prev_moves[prevCount][2] = {{0,2}, {0,3}, {0,4}, {1,1}, {4,4}, {2,2}, {2,3}, {3,1}, {3,2}, {2,4}, {-1,-1}};

            int next_moves[nextCount][2];

            int i =0;

            battle_ship(map, prev_moves, next_moves);

    // Content of the next_moves is {{1,0}, {3,0}, {3,4}, {-1,-1}}

}
```