Laboratory Work #5 - Pointers and dynamically memory allocation

Due Date: to be updated
Handover Date: Saturday, September 23, 2023

1 Introduction

In this laboratory work, we delve into the dynamic creation and manipulation of two-dimensional arrays using pointers and dynamic memory allocation. Unlike traditional static arrays, dynamic arrays allow us to allocate memory at runtime, providing flexibility in memory usage. We will explore how to dynamically allocate memory for 1D and 2D array, manage its rows and columns, and utilize pointers to access and manipulate individual elements efficiently. This hands-on experience will enhance your understanding of memory management in 1D and 2D arrays, a crucial skill for optimizing memory usage in various programming tasks.

2 Input

You will have to read initially 3 variables from console, k(1D array length), n and m(2D array size), allocate the memory then read from console the arrays. This time n should be equal to m
More info on memory allocation can be found here
!!!IMPORTANT - No fixed size arrays or hard-coded inputs, ALL THE ARRAYS SHOULD BE READ FROM CONSOLE

3 Task

Choose one/multiple tasks from the list below and build up your mark. To allocate/deallocate memory you should use malloc()/free() functions.

3.1 Piece of cake - each of them is worth 0.5 points

- implement a function that declares an integer variable, assigns it a value, and then uses a pointer to print the value of that variable.
- implement a function that swaps the values of two integer variables using pointers.

3.2 Easy - each of them is worth 1 point

- implement a function that receives the 1D array of integers and returns the largest and smallest values in the array using pointers.
- implement a function that accepts the 1D array array of integers and returns the sum of all negative numbers in the array using pointer arithmetic.
- implement a function that reverses the elements of the 1D integer array using pointers
- implement a function that takes the 2D integer array as input and transposes it (rows become columns, and vice versa) in-place using pointers

3.3 Medium - each of them is worth 2 points

- implement a function that accepts the 1D integer array and returns the index of the first occurrence of a specific value(to be read from console when the function is called). Use pointers for the search.

- implement a function that receives the 1D array of integers and a target value. The function should count how many times the target value(to be read from console when the function is called) appears in the array using pointers.
- implement a function that takes two 1D arrays(second array should be the original array but reversed) of integers and concatenates them into a new array using pointers and store the result in a new array which also has dynamically allocated memory
- implement a function that transposes the 2D array in-place(without using additional memory). The function should take the square matrix and its size as input and perform the matrix transposition operation using pointers.
- implement a function that takes the 2D array of integers along with its dimensions (n and m) as input. The function should calculate and return the sums of each row and each column in the array using pointers.

3.4 Hard - Implement a Stack Data Structure with Dynamic Memory Allocation AKA implement your own stack(worth 10 points)

Implement your own stack data structure in C using dynamic memory allocation and pointers. A stack is a linear data structure that follows the Last-In, First-Out (LIFO) principle. Your implementation should provide the following functionalities:

- Push: Add an element to the top of the stack.
- Pop: Remove and return the element from the top of the stack.
- Peek: Return the element at the top of the stack without removing it.
- IsEmpty: Check if the stack is empty.
- IsFull: Check if the stack is full (if there is a maximum capacity, but there should be one so choose one properly).

Task Requirements:

- Implement a stack data structure using dynamic memory allocation (malloc and free) and pointers to store the elements.
- Ensure that the stack can hold elements of any data type (integers, characters, floats or optionally user-defined structs).
- Define appropriate data structures (e.g. linked lists or arrays) to store the elements in the stack.
- Properly manage memory allocation and deallocation when pushing and popping elements from the stack.
- Ensure proper error handling, such as checking for stack underflow (trying to pop from an empty stack).
- Provide a demonstration in the main program that showcases the stack's functionality. For example, you can push and pop elements to/from the stack and display its contents.
- Test your stack implementation with various scenarios to ensure it functions correctly and handles different cases gracefully.

.

Example output:

Listing 1: Stack Operations Example

```
Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. IsEmpty
5. IsFull
6. Exit

Enter your choice: 1
Enter the value to push onto the stack: 42
Element 42 pushed onto the stack.

Stack Contents: 42

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. IsEmpty
5. IsFull
6. Exit

Enter your choice: 2
Popped element: 42

Stack Contents: (empty)

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. IsEmpty
5. IsFull
6. Exit

Enter your choice: 4
The stack is empty.

Stack Operations Menu:
1. Push
2. Pop
3. Peek
4. IsEmpty
5. IsFull
6. Exit

Enter your choice: 6
Exiting the program.
```

4 Grading

In order to get a mark for the rest of difficulties, you will have first of all to implement the task(base task) from Kulev_PC_SDA.pdf matching your variant(your number in the list). The base task is worth 5

points, so if you for only it, you will be marked only with 5.

You have the freedom of choice to build up your mark, however you must choose at least one medium problem. You should have at least 2 easy(or medium) problems that operate 1D and 2D arrays each.

If you choose the hardest difficulty, you will have 10 by default(no need for base task).

PS: No codeforces bonus for this one
PS2: For any questions, dont be shy to reach out to me or your colleagues

5 Reporting

You should run your program at least twice and include screenshots in the report, both times you should use different inputs and array length/size(does not apply to the hard problem)

IMPORTANT!!! As i specified on else, you have to use latex to create/format your report