Assignment no. 1

Title: Basics of Pointers, Structures, functions and DMA

- Part 1
    1. Declare and Initialize Pointers:
        a. Declare an integer variable named "num" and initialize it with a value of 20.
        b. Declare a pointer variable named "ptr" of type integer and assign it the address of the variable "num".
    2. Access and Modify Values Using Pointers:
        a. Print the value of "num" and the value pointed to by "ptr".
        b. Modify the value of "num" using the pointer "ptr" to assign a new value.
        c. Print the updated value of "num" and the value pointed to by "ptr".
    3. Pointer Arithmetic:
        a. Declare an array of integers named "arr" with 5 elements and initialize it with some values.
        b. Declare a pointer variable named "arrPtr" and assign it the address of the first element of the array.
        c. Using pointer arithmetic, access and print the values of all the elements in the array using "arrPtr".
    4. Pointer to Pointers:
        a. Declare two integer variables, "a" and "b", and assign them some values.
        b. Declare two integer pointers, "ptr1" and "ptr2", and assign them the addresses of "a" and "b" respectively.
        c. Declare a pointer to a pointer named "pptr" and assign it the address of "ptr1".
        d. Using the pointer to pointer, print the values of "a" and "b".
        e. Modify the value of "b" using the pointer to pointer.
        f. Print the updated value of "b" and the value of "a" using the pointers "ptr2" and "ptr1" respectively.

- Part 2
    1. Define a Structure:
        a. Declare a structure named "Employee" with the following members:
            - "name" (string) to store the employee's name
            - "id" (integer) to store the employee's ID number
            - "salary" (float) to store the employee's salary

        b. Declare a variable "emp1" of type "Employee" and initialize its members with sample data.
    2. Accessing Structure Members:
        a. Print the values of the "name," "id," and "salary" members of "emp1" using dot notation.
        b. Prompt the user to enter values for the members of "emp1" and store them using dot notation.
        c. Print the updated values of "emp1" members.
    3. Array of Structures:
        a. Declare an array of "Employee" structures named "employeeList" with a size of 3.
        b. Prompt the user to enter values for the members of each employee in the "employeeList" array.
        c. Print the details of each employee using a loop and dot notation.
    4. Nested Structures:
        a. Define a structure named "Date" with members "day" (integer), "month" (integer), and "year" (integer).
        b. Modify the "Employee" structure from earlier to include a "joiningDate" member of type "Date".
        c. Prompt the user to enter values for the "joiningDate" member of "emp1" and print the updated details.
    5. Passing Structures to Functions:
        a. Create a function named "printEmployeeDetails" that takes an "Employee" structure as a parameter and prints its details.
        b. Call the function "printEmployeeDetails" and pass "emp1" as an argument.

- Part 3
    1. Basic Function Creation and Calling:
        a. Create a function named "greet" that takes no parameters and prints a greeting message, such as "Hello, welcome to the lab!"
        b. Call the "greet" function from the main function to display the greeting message.
    2. Function with Parameters and Return Value:
        a. Create a function named "addNumbers" that takes two integers as parameters and returns their sum.
        b. Prompt the user to enter two numbers and store them in variables.
        c. Call the "addNumbers" function with the user-entered numbers as arguments and print the sum.
    3. Recursive Function:
        a. Create a recursive function named "factorial" that calculates the factorial of a given positive integer.
        b. Prompt the user to enter a positive integer and store it in a variable.
        c. Call the "factorial" function with the user-entered number as an argument and print the result.
    4. Function with Arrays:
        a. Create a function named "findMax" that takes an integer array and its size as parameters.
        b. Inside the function, find the maximum value in the array and return it.
        c. Declare an integer array and initialize it with some values.
        d. Call the "findMax" function with the array and its size as arguments and print the maximum value.

- Part 4
    1. Dynamic Memory Allocation - Single Variable:
        a. Prompt the user to enter an integer value.
        b. Allocate memory dynamically to store the entered integer using the appropriate function.
        c. Assign the entered value to the dynamically allocated memory.
        d. Print the value stored in the dynamically allocated memory.
        e. Deallocate the dynamically allocated memory.
    2. Dynamic Memory Allocation - Array:
        a. Prompt the user to enter the size of an integer array.
        b. Allocate memory dynamically to store the integer array of the specified size.
        c. Prompt the user to enter values for each element of the dynamically allocated array.
        d. Print the values stored in the dynamically allocated array.
        e. Deallocate the dynamically allocated memory.
    3. Dynamic Memory Allocation - String:
        a. Prompt the user to enter a string.
        b. Allocate memory dynamically to store the entered string using the appropriate function.
        c. Copy the entered string to the dynamically allocated memory.
        d. Print the string stored in the dynamically allocated memory.
        e. Deallocate the dynamically allocated memory.
    4. Dynamic Memory Allocation - Structure:
        a. Define a structure named "Student" with members "name" (string) and "age" (integer).
        b. Prompt the user to enter the name and age of a student.
        c. Allocate memory dynamically to store a "Student" structure.
        d. Assign the entered values to the dynamically allocated structure.
        e. Print the details of the student stored in the dynamically allocated structure.
        f. Deallocate the dynamically allocated memory.
    5. Dynamic Memory Allocation - 2D Array:
        a. Prompt the user to enter the number of rows and columns for a 2D integer array.
        b. Allocate memory dynamically to store the 2D integer array of the specified size.
        c. Prompt the user to enter values for each element of the dynamically allocated 2D array.
        d. Print the values stored in the dynamically allocated 2D array.
        e. Deallocate the dynamically allocated memory.

- Part 5
    1. Pass by Value vs. Pass by Address:
        a. Create a function named "changeValue" that takes an integer parameter.
        b. Inside the function, change the value of the parameter to a new value.
        c. Call the "changeValue" function from the main function and pass an integer variable as an argument.
        d. Print the value of the variable before and after the function call to observe the effect of pass by value.
    2. Pass by Value vs. Pass by Address with Arrays:
        a. Create a function named "modifyArray" that takes an integer array parameter and its size.
        b. Inside the function, modify the values of the array elements by adding 1 to each element.
        c. Call the "modifyArray" function from the main function and pass an integer array as an argument along with its size.
        d. Print the values of the array before and after the function call to observe the effect of pass by value.
    3. Pass by Address and Dynamic Memory Allocation:
        a. Create a function named "doubleValue" that takes an integer pointer parameter.
        b. Inside the function, double the value of the integer by dereferencing the pointer.
        c. Allocate memory dynamically for an integer variable in the main function.
        d. Call the "doubleValue" function from the main function and pass the address of the dynamically allocated integer.
        e. Print the value of the integer before and after the function call to observe the effect of pass by address.
    4. Pass by Value and Structures:
        a. Create a structure named "Person" with members "name" (string) and "age" (integer).
        b. Create a function named "changePersonAge" that takes a "Person" structure as a parameter and modifies its "age" member.
        c. Call the "changePersonAge" function from the main function and pass a "Person" structure as an argument.
        d. Print the values of the "age" member before and after the function call to observe the effect of pass by value.

- Part 6
    1. Swap Function Using Pointers: Implement a function that takes two integer pointers as parameters and swaps the values of the integers they point to. Test the function by swapping the values of two variables.
    2. Pointer Arithmetic with Structures: Declare a structure named "Student" with members "name" (string) and "age" (integer). Create an array of structures, allocate memory dynamically, and use pointer arithmetic to access and modify the values of the structure members.
    3. Pointers and Functions: Create a function that takes an integer array as a parameter and returns a pointer to the maximum element in the array. Test the function by passing an array and printing the maximum value using the returned pointer.
    4. Function with Strings: Create a function named "countVowels" that takes a string as a parameter and returns the number of vowels in the string. Test the function by passing different strings and printing the vowel count.
    5. Recursive Function with Strings: Create a recursive function named "reverseString" that takes a string as a parameter and reverses it. Test the function by passing different strings and printing the reversed strings.
    6. Dynamic Memory Allocation - Structure Array: Create a structure named "Book" with members "title" (string) and "author" (string). Prompt the user to enter the number of books and allocate memory dynamically to store an array of "Book" structures. Prompt the user to enter details for each book and print them.
    7. Dynamic Memory Allocation - Resize Array: Create a function that takes an integer array and its current size as parameters. Inside the function, reallocate memory to resize the array to double its size. Test the function by resizing an array and printing its contents.
    8. Dynamic Memory Allocation - Matrix Operations: Implement functions to perform matrix addition, multiplication, and transpose operations using dynamically allocated memory for matrices. Test the functions by performing matrix operations on user-entered matrices.