Lab Experiment- C Language

Objective:

- To gain practical experience with advanced pointer concepts in C, including pointer arithmetic, pointers and arrays, and function pointers. Master usage of gcc compiler and gdb.

Materials needed:

- Computer with a C compiler (e.g., GCC)
- Debugger (GDB)
- Text editor or IDE

Part 1: Pointer Basics and Arithmetic

Use this template code for this part onwards: template_code_Part1.c

Task 1.1

Create a program that demonstrates basic pointer usage:

- Declare an integer variable and a pointer to it
- Print the value of the variable using both direct access and the pointer
- Modify the value using the pointer and print the new value

Task 1.2

Implement a function that swaps two integers using pointers

Task 1.3

Create an array of integers and use pointer arithmetic to:

- Print all elements of the array
- Calculate the sum of all elements
- Reverse the array in-place

Part 2: Pointers and Arrays

Task 2.1

Create a 2D array (matrix) of integers and write functions to:

- Initialize the matrix with random values
- Print the matrix
- Find the maximum element in the matrix

Task 2.2

Implement a function that takes a 2D array as a parameter and calculates the sum of each row

Part 3: Function Pointers

Task 3.1

Create an array of integers and implement the following sorting algorithms:

- Bubble sort
- Selection sort

Task 3.2

Create a function pointer for the sorting algorithm and use it to sort the array

Task 3.3

Implement a simple calculator program that uses function pointers to perform basic arithmetic operations (addition, subtraction, multiplication, division)

Part 4: Advanced Challenge

Task 4.1

Implement a simple linked list with the following operations:

- Insert a node at the beginning
- Delete a node by value
- Print the list

Task 4.2

Use function pointers to create a generic linked list that can store different data types.

Part 5: Dynamic Memory Allocation

Task 5.1

Create a program that:

- Dynamically allocates an array of integers
- Allows the user to input the size of the array and its elements
- Calculates and prints the sum and average of the elements

Task 5.2

Implement a function that uses realloc() to extend an existing dynamically allocated array

Task 5.3

Create a simple memory leak detector:

- Write functions to allocate and free memory
- Keep track of allocated memory addresses
- Print a warning if the program ends with unfreed memory

Part 6: Structures and Unions

Task 6.1

Create a structure to represent a student with fields for name, ID, and grades in three subjects

Task 6.2

Implement functions to:

- Input student data

- Calculate the average grade
- Print student information

Task 6.3

Create a nested structure to represent a university with departments and students

Task 6.4

Implement a union to store different types of data (int, float, char) and demonstrate its usage

Part 7: File I/O

Task 7.1

Create a program that:

- Writes the student data (from Part 6) to a text file
- Reads the data back from the file and prints it

Task 7.2

Modify the program to write and read the student data in binary format

Task 7.3

Implement a simple log file system:

- Create functions to log messages with timestamps
- Allow appending to an existing log file
- Implement a function to read and display the log

Final Tasks:

Task Y: Sequential Multiplier in C with Unit Tests

- Write C code to implement Booth's multiplication algorithm. Write different functions for shifting and adding so that you can later visualize functions call stack.
- The function should take two signed integers as input and return their product.
- Use bit manipulation operators for efficient multiplication.
- Write a test function to verify the correctness of your Booth multiplier function.
- Create test cases for various scenarios, including positive, negative, zero inputs, multiplication by zero, multiplication by 1, and edge cases (e.g., overflow).

Task Z: Memory Management Maze (Pointers and Dynamic Allocation)

- Create a function to dynamically allocate memory for a 2D array representing the maze (walls and paths).
    - We'll represent the maze as a 2D array. However, instead of a static array declaration, we'll use dynamic allocation with pointers.
    - Define an int variable named **maze_size** to store the size of the maze (number of rows and columns).
- Use pointers to access and modify elements within the maze.
    - Use malloc to allocate memory for an array of pointers. Each pointer will point to a row in the maze.
    - For each row pointer, use another malloc to allocate memory for an array of ints representing the actual maze elements (walls = 1, paths = 0).
- Implement a function to navigate the maze using a pointer to the current position.
    - This function will take a pointer to the current position (***int* current_position***) and a pointer to the maze (***int** maze***) as arguments.
    - Use the current position's indices within the maze array to access the current element.
    - Check if the current element is a wall (value = 1). If so, the function should return a value indicating a dead end.
    - If the current element is a path (value = 0), check for the exit condition (e.g., reaching the last row and column). If it's the exit, return a value indicating success.
    - Otherwise, explore adjacent positions (up, down, left, right) using pointer arithmetic. Make sure to stay within the maze boundaries.
    - For each valid adjacent position, call the navigation function recursively, passing the new position pointer.

    - If the recursive call returns a success (found the exit), return success from the current function call.
    - If none of the adjacent positions lead to the exit, return a value indicating a dead end.
- Free up the allocated memory after navigation.
    - After using the maze, it's crucial to deallocate the dynamically allocated memory to avoid memory leaks.
    - Use free to free each row of the maze first, then, free the array of row pointers

Helping Material:

- Vim:
    - MIT OpenCourseware
- C Language:
    - The C Programming Language by Kernighan & Ritchie (available here)
    - C Tutorial – Tutorialspoint (available here)
- GCC:
    - Opensource: click here
    - IOFlood: click here
    - GNU GCC Guide: click here
- GDB:
    - Geeks for Geeks: click here
    - Medium: click here
    - Baeldung: click here
    - How to forge: click here