Open Ended Lab -1

Spring 2025

Course Title: Data Structures Lab

Course Code: CSE1302

Section: 02

Submitted by:

Student Name: Sakib Al Hasan

ID: 241014134

Department of CSE

University of Liberal Arts Bangladesh (ULAB)

Console-Based CRUD Application

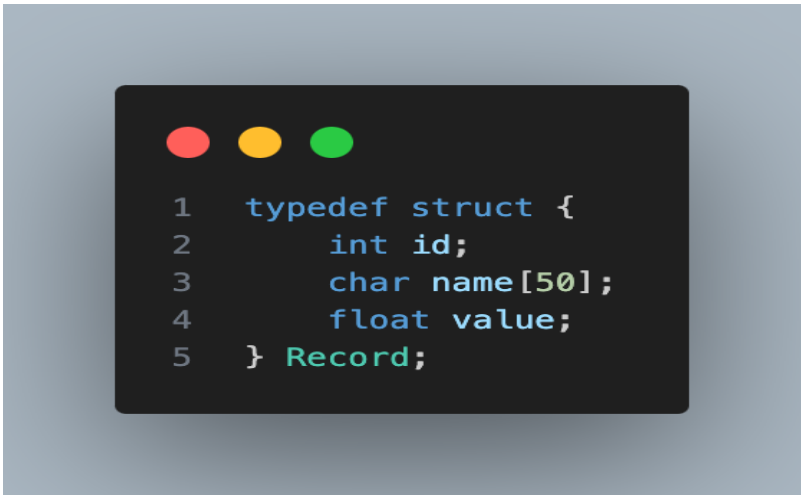
Objective:

The purpose of this experiment is to design and implement a simple **console-based CRUD (Create, Read, Update, Delete)** application using the C programming language. The program manages a list of records, each containing a unique ID, a name, and a numeric value. The system utilizes **dynamic memory allocation** to handle records efficiently during runtime.

Explanation of Data Structures Used

Data Structure:

The system uses a **dynamically allocated array of structures** to store all records. Each record is represented using the following structure:



```c
typedef struct {
    int id;
    char name[50];
    float value;
} Record;
```

Dynamic Memory Allocation:

- Records are stored using `malloc()` and `realloc()` to dynamically allocate memory as new records are added.
- `free()` is used to release memory after records are deleted or when the program exits.

This approach allows the program to manage memory efficiently and scale with user input during execution.

Full CRUD Implementation and Sample Output :

Create (Add a Record):

```c
void addRecord(Record **records, int *size) {
    *records = realloc(*records, (*size + 1) * sizeof(Record));
    if (*records == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    printf("Enter ID: ");
    scanf("%d", &(*records)[*size].id);
    printf("Enter Name: ");
    getchar();
    fgets((*records)[*size].name, 50, stdin);
    (*records)[*size].name[strcspn((*records)[*size].name, "\n")] = '\0';
    printf("Enter Value (e.g., GPA): ");
    scanf("%f", &(*records)[*size].value);

    (*size)++;
    printf("Record added successfully.\n");
}
```

Output:

```
m2air — crud — crud — 80×24
Last login: Wed Apr 16 17:29:48 on ttys003
/Users/m2air/Documents/University\ All/crud ; exi
m2air@m2s-MacBook-Air ~ % /Users/m2air/Documents/

====== CRUD Menu ======
1. Add Record
2. Display Records
3. Update Record
4. Delete Record
5. Search Record
6. Exit
Enter your choice: 1
Enter ID: 101
Enter Name: Sakib
Enter Value (e.g., GPA): 3.50
Record added successfully.
```

Read (Display All Records):

```c
void displayRecords(Record *records, int size) {
    printf("\n--- All Records ---\n");
    for (int i = 0; i < size; i++) {
        printf("ID: %d | Name: %s | Value: %.2f\n", records[i].id, records[i].name, records[i].value);
    }
}
```

Output:

```
====== CRUD Menu ======
1. Add Record
2. Display Records
3. Update Record
4. Delete Record
5. Search Record
6. Exit
Enter your choice: 2

--- All Records ---
ID: 101 | Name: Sakib | Value: 3.50
```

Update (Modify a Record by ID):

```c
void updateRecord(Record *records, int size) {
    int id;
    printf("Enter ID to update: ");
    scanf("%d", &id);

    for (int i = 0; i < size; i++) {
        if (records[i].id == id) {
            printf("Enter new Name: ");
            getchar();
            fgets(records[i].name, 50, stdin);
            records[i].name[strcspn(records[i].name, "\n")] = '\0';
            printf("Enter new Value: ");
            scanf("%f", &records[i].value);
            printf("Record updated successfully.\n");
            return;
        }
    }
    printf("Record with ID %d not found.\n", id);
}
```

Output:

```
====== CRUD Menu ======
1. Add Record
2. Display Records
3. Update Record
4. Delete Record
5. Search Record
6. Exit
Enter your choice: 3
Enter ID to update: 101
Enter new Name: Nupor
Enter new Value: 3.80
Record updated successfully.
```

Search (Find a Record by ID):

```c
void searchRecord(Record *records, int size) {
    int id;
    printf("Enter ID to search: ");
    scanf("%d", &id);

    for (int i = 0; i < size; i++) {
        if (records[i].id == id) {
            printf("ID: %d | Name: %s | Value: %.2f\n", records[i].id, records[i].name, records[i].value);
            return;
        }
    }
    printf("Record with ID %d not found.\n", id);
}

```

Output:

```
====== CRUD Menu ======
1. Add Record
2. Display Records
3. Update Record
4. Delete Record
5. Search Record
6. Exit
Enter your choice: 5
Enter ID to search: 101
ID: 101 | Name: Nupor | Value: 3.80
```

Delete (Remove a Record by ID):

```c
void deleteRecord(Record **records, int *size) {
    int id;
    printf("Enter ID to delete: ");
    scanf("%d", &id);

    for (int i = 0; i < *size; i++) {
        if ((*records)[i].id == id) {
            for (int j = i; j < *size - 1; j++) {
                (*records)[j] = (*records)[j + 1];
            }
            *records = realloc(*records, (*size - 1) * sizeof(Record));
            (*size)--;
            printf("Record deleted successfully.\n");
            return;
        }
    }
    printf("Record with ID %d not found.\n", id);
}

```

Output:

```
====== CRUD Menu ======
1. Add Record
2. Display Records
3. Update Record
4. Delete Record
5. Search Record
6. Exit
Enter your choice: 4
Enter ID to delete: 101
Record deleted successfully.
```

Sample Main Menu

```c
int main() {
    Record *records = NULL;
    int size = 0, choice;

    do {
        printf("\n====== CRUD Menu ======\n");
        printf("1. Add Record\n");
        printf("2. Display Records\n");
        printf("3. Update Record\n");
        printf("4. Delete Record\n");
        printf("5. Search Record\n");
        printf("6. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: addRecord(&records, &size); break;
            case 2: displayRecords(records, size); break;
            case 3: updateRecord(records, size); break;
            case 4: deleteRecord(&records, &size); break;
            case 5: searchRecord(records, size); break;
            case 6: printf("Exiting program. Goodbye!\n"); break;
            default: printf("Invalid choice. Please try again.\n");
        }

    } while (choice != 6);

    free(records);
    return 0;
}
```

Lab Report Summary

This project demonstrates the use of **dynamic memory allocation** and **arrays of structures** to manage data in real-time. All basic CRUD operations were implemented and tested through a user-friendly, menu-driven interface.

Key Features:

- Dynamic memory allocation using `malloc()` and `realloc()`
- Clean modular functions for each operation
- Safe and interactive record management

Conclusion

The Console-Based CRUD Application successfully illustrates how core programming concepts like **structures**, **dynamic memory**, and **modular design** can be applied to solve real-world problems. This project improves understanding of memory management, user input handling, and data processing in C.