COS 201 Computer programming Mid Semester Assessment Group Project

Student Record System - Project Presentation

Project title: Student Record System

Project Overview

Assessment Exercise

Create a simple Student Record System in C that allows users to manage student information.

Our Solution

Our group came together to build a Menu Driven Application that handles all core student management Operations with good and capable Storage Capabilities

Our Group Project Objectives

- Create a functional student management system
- Show proficiency in C programming concepts
- Showcase the software engineering principles that were taught to us during the semester
- Provide an above average user experience

System Architecture -Supervised by (Jasry Egbowon)

Program Structure

```
MAIN PROGRAM
    │
    ├── Welcome & User Authentication
    ├── Main Menu System
    │
```

```
├── Student Management Module
│   ├── Add Student Records
│   ├── Display All Students
│   ├── Modify Existing Records
│   └── Search Functionality
│
├── Academic Operations
│   ├── Pass/Fail Evaluation
│   ├── Average Marks Calculation
│   └── Sorting by Performance
│
└── Data Persistence
    ├── Save to File
    └── Load from File
```

Data Flow Diagram

```
User Input → Menu Processing → Function Execution → Data Manipulation → Output Displa
                          ↓
              File Storage/Retrieval
```

Core Implementation based on what we learnt this semester

Data Structure Design

```c
typedef struct {
    char name[50];
    int rollNumber;
    float marks;
} Student;
```

We used structs to create custom data types for organizing student information

System Configuration

```c
#define MAX_STUDENTS 100
#define PASS_MARK 40.0
```

```
Student students[MAX_STUDENTS];
int studentCount = 0;
char currentUser[50];
```

Week 4- COS201- Computer programming: ARRAYS & CONSTANTS - We fixed-size arrays for data storage and macro definitions for system limits

---

Application Feature Overview & Implementation

1. User Interface & Welcome System

We Implemented: User input and output

```
//This is our intro
void displayWelcome() {
    printf("--------------------------------\n");
    printf("   STUDENT RECORD SYSTEM\n");
    printf("--------------------------------\n");
    printf("Welcome! Please enter your name: ");
    scanf("%s", currentUser);
    printf("Hello, %s! Let's manage some student records.\n", currentUser);// A
pointer is used to get the name gotten from the input name
}
```

2. Main Menu System

We implemented: Continuous loop with switch-case structure

```
// Since no GUI- Graphic user interface was used, we tried to use a simple Menu-
driven loop for the Student Record Systems interface
do {
    // This block is the main menu
    printf("\n--- Student Record System ---\n");
    printf("1. Add New Student\n");
    printf("2. Display All Students\n");
    printf("3. Search Student by Roll Number\n");
    printf("4. Modify Student Record\n");
    printf("5. Calculate Average Marks\n");
    printf("6. Sort Students by Marks\n");
    printf("7. Save Records to File\n");
    printf("8. Load Records from File\n");
    printf("9. Exit\n");
    printf("Enter your choice: ");

    // The program reads the user's choice
    scanf("%d", &choice);

    //An action is made based on the user's choice
    switch(choice) {
        case 1:
            addStudent();  // Add a new student
            break;
        case 2:
            displayAllStudents();  // Display all students
            break;
        // there are other cases used: search, modify, calculate average, sort,
save, load
        // case 3: searchStudent(); break;
        // case 4: modifyStudent(); break;
        // case 5: calculateAverage(); break;
        // case 6: sortStudents(); break;
        // case 7: saveToFile(); break;
        // case 8: loadFromFile(); break;
        // case 9: break; // Exit
        default:
            printf("Invalid choice! Please try again.\n");
    }

} while(choice != 9);  // it Repeats until the user chooses to exit
```
Week4: Lab4- Loops and Iterations : DO-WHILE LOOPS & SWITCH-CASE - We created an iterative menu system with conditional branching
We also made use of control flow which we were also taught in the conditions used

3. Student Addition Module

We Implemented: Input/Output validation along with array management

```
// A Function to add a new student to the array
```

```
void addStudent() {
    // The program Checks if we have reached the maximum allowed number of
students.
    if(studentCount >= MAX_STUDENTS) {
        printf("Sorry, maximum capacity reached!\n");
        return;  //The program Exits the function if no more students can be
added
    }

    Student newStudent;  //it was my idea to create a Temporary variable to store
the new student's details

    // Prompt the user to enter the student's name
    printf("Enter student name: ");
    scanf("%s", newStudent.name);  // Read name (note: no spaces allowed)

    // Prompt the user to enter the student's roll number
    printf("Enter roll number: ");
    scanf("%d", &newStudent.rollNumber);

    // Prompt the user to enter the student's marks
    printf("Enter marks: ");
    scanf("%f", &newStudent.marks);

    // Add the new student to the students array
    students[studentCount] = newStudent;

    // a simple Increment statement is used to increase the student count one by
one
    studentCount++;
}
```

4. Display All Students

We Implemented: Tabular data presentation with pass/fail status

```
// Function to display all student records in a tabular format
void displayAllStudents() {
    // Print the header for the table
    printf("\n--- All Student Records ---\n");
    printf("%-5s %-20s %-10s %-10s\n", "S.No", "Name", "Roll No", "Marks");
    printf("-------------------------------------------------\n");

    // Loop through each student and display their details
    for(int i = 0; i < studentCount; i++) {
        char status[10];  // To store PASS or FAIL status

        // A loop Determines whether the student has passed or failed
        if(students[i].marks >= PASS_MARK) {
            strcpy(status, "PASS");  // Copy "PASS" into status
        } else {
            strcpy(status, "FAIL");  // Copy "FAIL" into status
        }

        // Print student details in a formatted row
        // %-5d  : Serial number, left-aligned, width 5
        // %-20s : Name, left-aligned, width 20
        // %-10d : Roll number, left-aligned, width 10
        // %-10.2f : Marks, left-aligned, width 10, 2 decimal places
        // (%s) : PASS/FAIL status
        printf("%-5d %-20s %-10d %-10.2f (%s)\n",
               i+1, students[i].name, students[i].rollNumber,
               students[i].marks, status);
    }
}
```

Week4: Lab4- Loops and Iterations- Demonstrate iterative data processing and decision making

5. Search Functionality

We implemented: Item-by-item search

```
// Function used to search for a student by their roll number
void searchStudent() {
    int rollNumber;

    // Ask the user to enter the roll number to search for
    printf("Enter roll number to search: ");
    scanf("%d", &rollNumber);
```

```c
    // Call helper function to find the index of the student in the array
    int index = findStudentByRoll(rollNumber);

    // If the student is not found, inform the user
    if(index == -1) {
        printf("Student not found!\n");
        return;  // Exit the function
    }

    // If found, display the student's details
    printf("Name: %s\n", students[index].name);
    printf("Roll Number: %d\n", students[index].rollNumber);
    printf("Marks: %.2f\n", students[index].marks);
}

// Helper function to find a student's index by roll number
// Returns the index of the student in the array if found, or -1 if not found
int findStudentByRoll(int rollNumber) {
    // Loop through all students
    for(int i = 0; i < studentCount; i++) {
        // Check if the current student's roll number matches the search
        if(students[i].rollNumber == rollNumber) {
            return i;  // Return the index if found
        }
    }
    return -1;  // Return -1 if student not found
}
```

*Week5: Methods COS201- Computer programming-
FUNCTIONS & PARAMETERS - We used methods to get return values

6. Average Calculation

We Implemented: Mathematical computation

```c
// Function to calculate and display the average marks of all students
void calculateAverage() {
    // Check if there are any students
    if(studentCount == 0) {
        printf("No students to calculate average.\n");
        return;  // Exit the function early if there are no students
    }

    float total = 0;  // Variable to store the sum of all marks

    // Loop through all students and add their marks to the total
    for(int i = 0; i < studentCount; i++) {
        total += students[i].marks;
    }

    // Calculate the average by dividing total marks by the number of students
    float average = total / studentCount;

    // Print the average marks rounded to 2 decimal places
    printf("Average marks: %.2f\n", average);
}
```

7. Sorting Algorithm

We implemented: Bubble sort for student records

```
// A Function we created to sort students based on their marks
void sortStudents() {
    int order;

    // The program Asks the user whether to sort in ascending or descending order
    printf("1. Ascending\n2. Descending\nChoice: ");
    scanf("%d", &order);

    // Bubble Sort algorithm
    // Outer loop controls the number of passes
    for(int i = 0; i < studentCount - 1; i++) {

        // Inner loop compares adjacent elements
        for(int j = 0; j < studentCount - i - 1; j++) {
            int shouldSwap = 0;  // This here is to determine if elements should
be swapped
```

```c
            // Decide whether to swap based on sorting order
            if(order == 1) {
                // Ascending order: swap if current marks > next marks
                shouldSwap = students[j].marks > students[j+1].marks;
            } else {
                // Descending order: swap if current marks < next marks
                shouldSwap = students[j].marks < students[j+1].marks;
            }

            // Perform the swap if needed
            if(shouldSwap) {
                Student temp = students[j];
                students[j] = students[j+1];
                students[j+1] = temp;
            }
        }
    }
}
```

[PLACEHOLDER: NESTED LOOPS & ALGORITHMS - Demonstrate sorting implementation and nested iterations]

8. File Operations

We implemented: Data persistence

```c
void saveToFile() {
// Function to save all student records to a file
void saveToFile() {
    char filename[50];  // Array to store the filename entered by the user

    // Ask user for the name of the file to save data into
    printf("Enter filename: ");
    scanf("%s", filename);  // Read the filename from user input

    // Open (or create) the file in write mode
    // "w" means any existing content will be overwritten
    FILE *file = fopen(filename, "w");

    // Write each student's data to the file
    // Loop through all student records using studentCount
    for(int i = 0; i < studentCount; i++) {
        fprintf(file, "%s %d %.2f\n",
                students[i].name,        // Write student's name
                students[i].rollNumber,  // Write student's roll number
                students[i].marks);      // Write student's marks (float with 2
decimals)
    }

    // Close the file after writing (important to avoid data loss)
    fclose(file);
}
// Function to load student records from a file
void loadFromFile() {
    char filename[50];  // Array to store the filename entered by the user

    // Ask the user for the name of the file to read from
    printf("Enter filename: ");
    scanf("%s", filename);  // Read the filename from user input

    // Open the file in read mode ("r")
    FILE *file = fopen(filename, "r");

    // Reset the student count before loading new data
    studentCount = 0;

    /* Read data line-by-line from the file.
       fscanf returns the number of items successfully read.
       We expect 3 items per line: name (string), roll number (int), marks
(float).
       The loop continues as long as fscanf successfully reads all 3 values. */
    while (fscanf(file, "%s %d %f",
                  students[studentCount].name,
                  &students[studentCount].rollNumber,
                  &students[studentCount].marks) == 3) {
        studentCount++;  // Move to the next student slot
    }

    // Close the file after loading all data
    fclose(file);
}
```
[PLACEHOLDER: FILE I/O - Explain file handling for data persistence]

Testing & Validation

Test Cases Covered

- Input validation and error handling
- Boundary conditions (empty list, full capacity)
- File operations (save/load consistency)
- Search functionality (existing/non-existing records)

Error Handling cases

- Invalid menu choices
- Duplicate roll numbers
- File access errors
- Array bounds checking

Here are the locations in the source where each of the above error-handling cases is implemented

- **Invalid menu choices**: handled in `main()` via the `switch` default case which prints an error message.

  Example (in `main()`):

  ```
  default:
          printf("Invalid choice! Please try again.\n");
  ```
- **Duplicate roll numbers**: handled in `addStudent()` by checking `findStudentByRoll(...)` before adding a new student.

  Example (in `addStudent()`):

  ```
  if(findStudentByRoll(newStudent.rollNumber) != -1) {
          printf("Error: Student with roll number %d already exists!\n",
  newStudent.rollNumber);
          return;
  }
  ```
- **File access errors**: handled in `saveToFile()` and `loadFromFile()` by checking the return value of `fopen()` and reporting an error when `NULL` is returned.

Examples:

```c
// saveToFile()
FILE *file = fopen(filename, "w");
if(file == NULL) {
        printf("Error creating file!\n");
        return;
}

// loadFromFile()
FILE *file = fopen(filename, "r");
if(file == NULL) {
        printf("Error: Could not open file %s\n", filename);
        return;
}
```

- **Array bounds checking / capacity checks**: handled in addStudent() (capacity check) and in loadFromFile() (reading loop limits by MAX_STUDENTS). Other functions also check for an empty list before operating (e.g., displayAllStudents(), modifyStudent()).

  Examples:

```c
// addStudent() does a capacity check to see if it has reached the maximum
number of students
if(studentCount >= MAX_STUDENTS) {
        printf("Sorry, we've reached the maximum number of students!\n");
        return;
}

// loadFromFile() ensures we don't overflow the array while reading
//because we are coding in C programming language we need to make sure of
things like these
while(studentCount < MAX_STUDENTS && fscanf(file, "%s %d %f",
            students[studentCount].name,
            &students[studentCount].rollNumber,
            &students[studentCount].marks) == 3) {
        studentCount++;
}

// Example checks for empty list before operations
if(studentCount == 0) {
        printf("No students in the system yet.\n");
```

```
            return;
    }
```

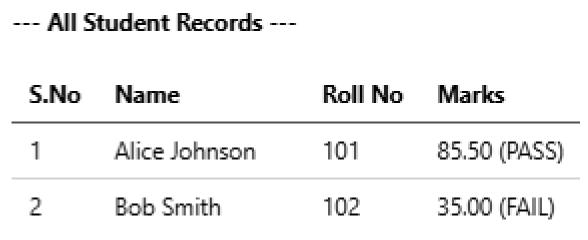
Program Output Samples

Sample Session Flow

```
---------------------------------
   STUDENT RECORD SYSTEM
---------------------------------
Welcome! Please enter your name: John
Hello, John! Let's manage some student records.

--- Student Record System ---
1. Add New Student
2. Display All Students
...
```

Data Display Format



--- All Student Records ---

S.No	Name	Roll No	Marks
1	Alice Johnson	101	85.50 (PASS)
2	Bob Smith	102	35.00 (FAIL)

Learning Outcomes from this project- what did we gain from it?

We gained:

Technical Skills:

- C programming proficiency
- Data structure implementation
- Algorithm design and optimization
- File handling techniques

- Memory management understanding

Software Concept mastery

- Software development lifecycle
- Problem-solving methodologies
- Code organization and modularity
- Debugging and testing strategies

Team Contribution

Team Members & Roles

1. **Egbowon Jasry Ayomikun**
    - Matric No:2024/B/SENG/0276
    - Student ID: 30115716

    Role:Lead Programmer & System Architect
2. **Elizabeth Bakare Eharomubo**
    - Matric No:2024/B/CSC/0314
    - Student ID: 30166148

    Role:Documentation & Testing
3. **Vivian Nkiruka Zoho**
    - Matric Number: 2024/B/IT/0088
    - Student ID:30165772
4. **Lawrence Paul**
    - Matric no: 2024/B/SENG/0313
    - Student ID: 30036428

    Role: Algorithm Optimization
5. **Abidoye Oluwatobi**
    - Matric No: 2024/B/SENG/0179
    - Student ID: 30111068

    Role: UI Design & Validation
6. **Nwachukwu Light Chidoziem**
    - Matric No: 2024/B/CSC/0241
    - Student ID:30110887

    Role:

Development Methodology

We used:

- Modular incremental development-
- Regular code reviews
- Comprehensive testing
- Documentation-driven approach

Acknowledgment of Mr. Emeka

We express our sincere gratitude to our lecturer for the guidance and opportunity to work on this comprehensive project. Even though there were a lot of issues coming up he showed up for us to mark our project for us, we thank you Mr. Emeka for that.