I. Introduction

- **Project Title:** EWU Student Portal System

- *Authors:*

1. Lajim, ID: 2022-3-60-043 - Role: Backend
2. Asif, ID: 2022-3-60-007 - Role: Frontend

Abstract: The EWU Student Portal is a Console-based platform designed to streamline student interactions with East West University's administrative services. This project aims to enhance user experience by providing easy access to academic resources, course registration, grades, and a comprehensive software solution designed to manage various aspects of university operations, including student records, course management, faculty data, and advising. The project aims to streamline university administrative tasks, improve data accuracy, and enhance user experience for students, faculty, and administrators.

II. Project Description

Problem Statement:

East West University lacks an integrated platform for students to access academic information, register for courses, and communicate with faculty and staff efficiently. This project addresses the need for a centralized student portal to facilitate these interactions.

Motivation:

Improving student access to academic resources and administrative services can enhance their overall learning experience and streamline university operations. The EWU Student Portal aims to bridge the gap between students and university services, promoting efficiency and convenience.

Scope and Limitations:

- **Scope**: The project covers features such as student registration, course enrollment, grade tracking, news updates, and communication with faculty.

- **Limitations**: The portal does not include features such as financial management or alumni services.

III. Methodology

C Language and Libraries Used:

- **C Language Version**: C99
- **Libraries**: Standard Input/output Library, String Library

Explanation:

Code Snippet: Checking addStudent Credentials

```
void addStudent (const char* studentName, const char* studentID, float
studentCgpa, float completedCredit, const char* password)
{
  struct Student* newStudent = (struct Student*)malloc(sizeof(struct Student));
   newStudent->studentName = strdup(studentName);
   newStudent->studentID = strdup(studentID);
   newStudent->studentCgpa = studentCgpa;
   newStudent->completedCredit = completedCredit;
   newStudent->password = strdup(password);
   newStudent->enrolledCourses = NULL;
   newStudent->next = students;
   students = newStudent;
}
```

- **Inputs**: Takes name, id, gpa, credits, and password as arguments.

- **Functionality**:

  Copies the provided details into the students array and increments the numStudents counter.

- **Purpose**: Adds a new student to the system.

- **Memory Allocation**: The function allocates memory for a new student structure using **malloc.** This memory allocation ensures that the new student structure has enough space to store all its attributes.

- **Initialization**:

  Each attribute of the new student structure is initialized:

  i. studentName and studentID are set to the values provided as arguments. The **strdup** function is used to duplicate the string values to ensure that they are safely stored in memory.

  ii. studentCgpa and completedCredit are set to the corresponding values provided as arguments.

  iii. password is set to the provided password value after duplicating it using **strdup**.

  iv. enrolledCourses is initialized to NULL, assuming that it will later point to a list of enrolled courses for the student.

  v. next is set to point to the existing list of students (students), effectively inserting the new student at the beginning of the list.

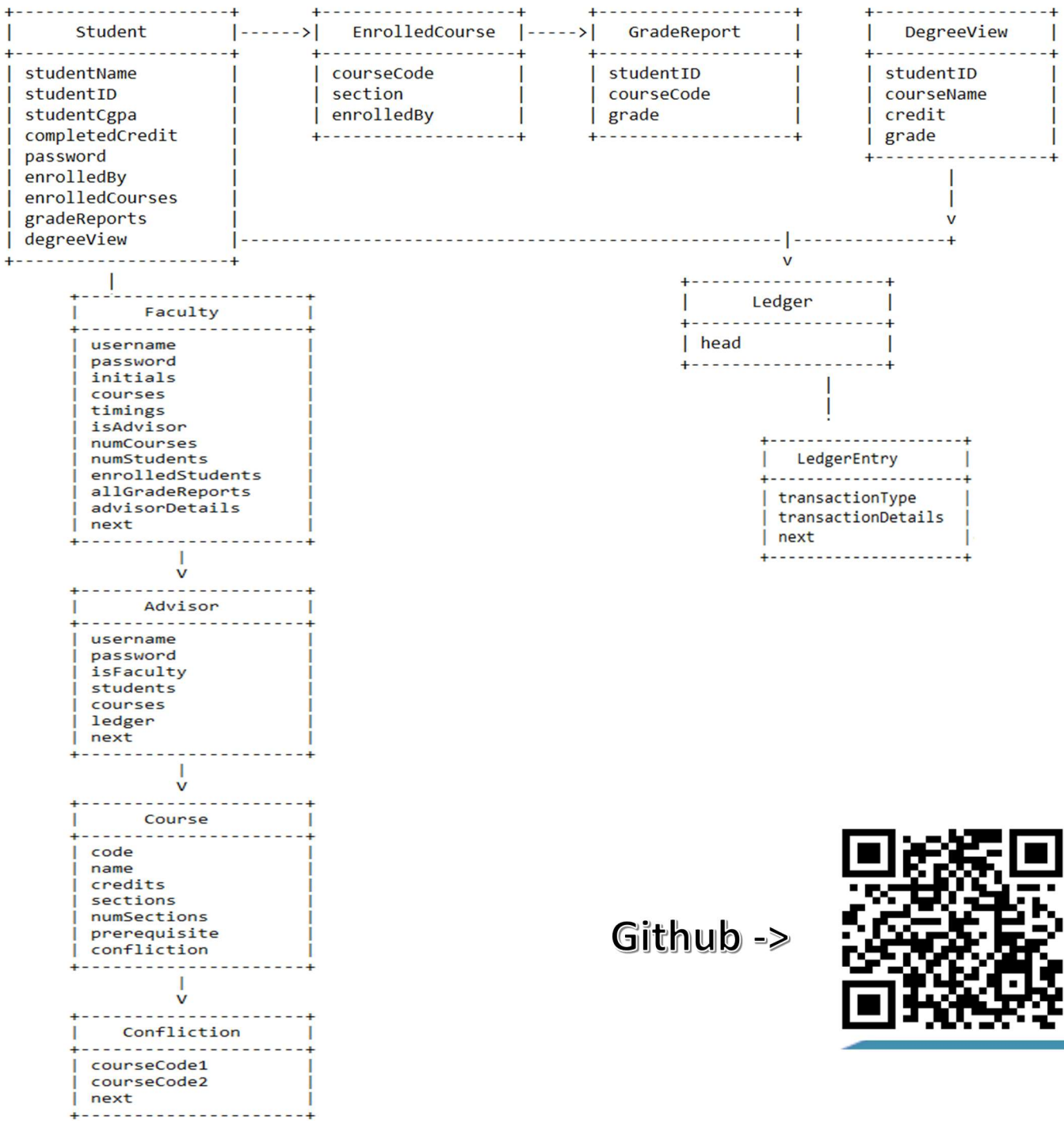
- **Updating Student List**:

  Finally, the students pointer is updated to point to the newly added student, making it the new head of the student list.

Algorithm and Data Structures:

- **Algorithm**: Linear Search for user authentication.
- **Data Structures**: Linked List,Stack,Tree,Queue,Arrays for storing user data.

Flowchart or Diagram:



Github ->

1. **Student**:
    - Connected to:
    - EnrolledCourse: Through the enrolledCourses pointer, which maintains a linked list of courses in which the student is enrolled.
    - GradeReport: Through the gradeReports pointer, which maintains a linked list of grade reports for the student.
    - DegreeView: Through the degreeView pointer, which maintains a linked list of the student's degree views.
    - Ledger: Not directly connected, but ledger entries might contain transactions related to students.

2. **Faculty**:
    - Connected to:
    - Student: Through the enrolledStudents pointer, which maintains a linked list of students advised by the faculty.
    - GradeReport: Through the allGradeReports pointer, which may contain grade reports of students advised by the faculty.
    - Advisor: Through the **advisorDetails** pointer, which contains information about the faculty's advisor role.

3. **Advisor**:
    - Connected to:
    - Student: Through the students pointer, which maintains a linked list of students advised by the advisor.
    - Ledger: Through the ledger pointer, which might contain transactions related to advisor activities.

4. **Course**:
    - Connected to:
    - EnrolledCourse: Courses are enrolled by students via the EnrolledCourse structure.
    - Confliction: Courses may have conflicts with other courses, which are represented through the Confliction structure.

5. **Confliction**:
    - Course: Through the courseCode1 and courseCode2 pointers, which represent conflicting courses.

6. **EnrolledCourse**:
    - Connected to:
    - Student: Through the enrolledBy field, which indicates the student who enrolled in the course.
    - Course: Through the courseCode field, which indicates the enrolled course.

7. **GradeReport**:
    Connected to:
    - Student: Through the studentID field, which indicates the student associated with the grade report.
    - Course: Through the courseCode field, which indicates the course associated with the grade report.

8. **DegreeView**:
    - Connected to:
    - Student: Through the studentID field, which indicates the student associated with the degree view.

9. **Ledger**:
    - Connected to:
    - Student: Indirectly, ledger entries may contain transactions related to student activities.
    - Faculty: Indirectly, ledger entries may contain transactions related to faculty activities.
    - Advisor: Indirectly, ledger entries may contain transactions related to advisor activities.
10. **LedgerEntry**:
    - Through the next pointer, forming a linked list of ledger entries.

Definitions:

1. **Course**: Stores course information such as code, name, credits, prerequisites, and sections. **Confliction:** Tracks conflicts between courses.
2. **Student**: Holds details like name, ID, CGPA, completed credits, enrolled courses, grade reports, and degree view.
3. **Advisor**: Contains information about academic advisors, including their username and password.
4. **Faculty**: Represents faculty members with their initials, taught courses, and schedules.
5. **EnrolledCourse**: Maintains records of student enrollment in courses, including course code, section, and advisor.
6. **GradeReport**: Keeps track of student grades, linking student IDs with course codes.
7. **DegreeView**: Provides a snapshot of a student's degree progress, including completed and remaining requirements.
8. **LedgerEntry**: Records transaction history including type and details.

Function Descriptions:

1. **Student Management** : Add, Update, Remove Student.
2. **Course Management** : Add, Update, Remove Course.
3. **Enrollment Management**: Enroll, Drop Student from Course.
4. **Grade Management** : Add, Update, View Grade Report.
5. **Advisor Management** : Add, Update, Remove Advisor.
6. **Faculty Management** : Add, Update, Remove Faculty.

Example Usage:

- **Adding a New Student**: Add a student with their details and verify correctness.
- **Enrolling a Student in a Course**: Select student and course, check prerequisites and conflicts, then enroll if conditions are met.
- **Generating a Grade Report**: Access student's academic record, compile grades, and display the report.

Implementation Details:

- ✓ Backend logic for authentication, course management, and grade handling.
- ✓ Simulated database functionality with file I/O.
- ✓ Command-line interface (CLI) for user interaction.
- ✓ Basic authentication using username/password.
- ✓ Course registration and grade management functions.
- ✓ Efficient data structures for storage and retrieval.
- ✓ Robust error handling and input validation.
- ✓ Thorough testing and debugging for reliability.

IV. Results and Discussion

- Testing and Validation:

  The project was tested extensively to ensure functionalities such as user authentication, course registration, and grade tracking work as expected. Test cases covered both normal and edge scenarios to validate robustness.

- Output and Performance:

  Sample outputs include successful user authentication, course registration, and grade display. The system performs efficiently with minimal delay in processing user requests.

- Evaluation and Analysis:

  The EWU Student Portal successfully meets its objectives of providing students with easy access to academic information and administrative services. However, future enhancements could include integration with additional university services and improved user interface design.

V. Conclusion

Summary:

The EWU Student Portal enhances student access to academic resources and administrative services at East West University. The project demonstrates the feasibility of implementing a centralized platform to streamline university operations.

Future Work:

Future enhancements could include integration with financial management systems, alumni services, and mobile application development for increased accessibility

------ The End ------