Data Structures Course Project - Fall 2024

University System

This project involves designing a university management system that efficiently handles student records, course information, enrollment histories, and course registrations using a variety of data structures. It integrates components such as linked lists, stacks, queues, binary search trees, and hash tables to ensure optimal performance for diverse operations.

System Overview:

1. **Student Records Management**: Use a Single Linked List (SLL) to store and manage student details.
2. **Course Records Management**: Implement a Binary Search Tree (BST) to organize and search course information.
3. **Course Enrollment History**: Employ a Double Linked List (DLL) to track each student's course history.
4. **Course Prerequisites**: Use a Stack to validate prerequisites for course registration.
5. **Course Waitlists**: Manage waitlists with a Queue.
6. **Search Optimization**: Implement Linear and Binary Search algorithms.
7. **Fast Access**: Utilize a Hash Table for efficient lookups.

Functional Components:

1. Student Records Management: Single Linked List

Store and manage student records in a Single Linked List.

Functionalities to implement:

- add(): Add a new student with attributes such as ID, Name, Email, Phone, Address, and Password.
- delete(): Remove a student by their ID.
- display(): Print all student details.

2. *Course Records Management: Binary Search Tree (BST)*

Store course information in a Binary Search Tree (BST).

Functionalities to implement:

- Each node contains: CourseID, CourseName, CourseCredits, and CourseInstructor.
- The tree is structured so that for any node, all nodes in the left subtree have smaller IDs, and all nodes in the right subtree have larger IDs.
- Methods:
  - addCourse()
  - dropCourse()

3. *Course Enrollment History: Double Linked List*

Track each student's course enrollment history using a Double Linked List.

Functionalities to implement:

- add(): Add a new enrollment record for a student.
- view(): Display the enrollment history of a specific student.

4. *Course Registration: Stack*

Validate course prerequisites using a Stack.

How it works:

- Push all required prerequisites onto the stack.
- Pop completed prerequisites from the stack based on the student's record.
- If the stack is empty, the student is eligible to register.

Functionalities to implement:

- validatePrerequisites(courseID, studentID): Check if a student meets the prerequisites for a course.
- Display remaining prerequisites if validation fails.

5. *Course Waitlist: Queue*

Manage course waitlists using a Queue.

Functionalities to implement:

- enqueue(): Add a student to the waitlist when a course is full.
- dequeue(): Enroll the first student on the waitlist when a spot becomes available.

6. *Search and Sort Operations*

Implement search and sort functionalities for:

- Students by ID.
- Courses by ID.

Bonus Part:

7. *Fast Access Using Hashing*

Enhance lookup efficiency with a Hash Table.

Requirements:

- Design a hashing function.
- Implement collision handling (e.g., Chaining or Open Addressing).
- searchWithHashing(): to improve course search efficiency.

Important Rules:

- This is a **teamwork project**, with 4 members at most, all must have the same TA, and if the team size exceeds 4, all team members will get zero.
- Handle **exception cases** throughout the project (Like divided over zero).
- All members must **understand** the code fully.
- Submit the project file with member IDs in the format: 20000000_20000000_20000000_20000000.cpp
- Grades are based on the discussion of your code.

- Cheating results in a grade of **ZERO**.

Grading Rubric:

Category	Points
Code Functionality	20%
Code Clarity	10%
Documentation	10%
Teamwork and Effort	10%
Discussion and Understanding	50%

Submission Deadline:

- **Deadline**: January 25, 2025, at 11:59 PM.
- **Submission**: Submit a single .zip file through Moodle.

Discussion Date: To Be Determined (TBD).