Technical Report: Final Project
EECE 2560: Fundamentals of Engineering Algorithms

Steve A. Nketiah and Kate Rober
Department of Electrical and Computer Engineering
Northeastern University
Nketiah.st@northeastern.edu
rober.k@northeastern.edu

December 5, 2024

Contents

1 Project Scope

The aim of this project is to design and develop a class registration system that allows students to register for seats in a class where the order of priority for registration depends on the student's grade level. It also facilitates the organization of students within the class based on alphabetical order. A wait-list will be included so that if a class is full, students will join a wait-list based on who joined it first. If a seat opens up, it will be filled by the wait-listed student first in line. The system will solve the problem of student registration and overfilling a course, which will improve the efficiency and student experience

The project's main objectives are:

- To allow students to efficiently register for courses.
- Have an easy-to-use menu interface
- Create a process for randomly filling seats for user simulation
- Record student and course information
- Use randomization to simulate the real world course registration systems.

The expected outcomes include a functioning C++ based program, proper documentation, and a final project report.

2 Project Plan

2.1 Timeline

The overall timeline for the project is divided into phases:

- **Week 1 (October 7 - October 13)**: Define project scope, establish team roles, and outline skills/tools.
- **Week 2 (October 14 - October 20)**: Begin development, set up the project repository, and start coding basic system functionalities.
- **Week 3 (October 21 - October 27)**: Continue coding, work on backend integration, and start writing technical documentation.
- **Week 4 (October 28 - November 3)**: Finish with the first draft of the backend code.
- **Week 5 (November 4 - November 10)**: Optimize backend code to make it more efficient. Start with frontend code.
- **Week 6 (November 11 - November 17)**: Revise and finalize the technical report and the PowerPoint presentation.
- **Week 7 (November 18 - November 28)**: Final presentation, report submission, and project closure.

2.2 Milestones

Key milestones include:

- Project Scope and Plan completed by end of Week 1.
- Initial development and setup of repository completed by Week 2.
- Core algorithm completion by Week 3.
- Final testing and project wrap-up by Week 4.

3 Team Roles

- **Steve Nketiah**: Responsible for making final edits to the main code, Pseudocode, time complexity analysis, and also report writing.
- **Kate Rober**: Responsible of implementation of overall structure of the algorithm, Writing test cases, and presentation.

4 Methodology

4.1 Pseudocode and Complexity Analysis

The following are the main functions:

- enrollStudent – This function enrolls student into the course
- insert – This function inserts a student node to a linked list in alphabetical order
- Drop student – This function removes a student from a course in a sorted manner
- remove – This function removes a student from the linked list. It returns true if the function has been successfully removed and false if it hasn't.
- display – This function displays all the students in a class
- FillCoursesRandomly – This function randomly creates courses and fills them up with students to simulate a real course registration system

Algorithm 1 Enroll Student in Course

```
procedure EnrollStudent(Student student)
    if enrolledCount < maxCapacity then
        enrolledStudents.insert(student)
        enrolledCount ← enrolledCount +1
        print "Enrollment successful for " student.firstName, student.lastName
    else if waitlist.size() < maxWaitlistCapacity then
        waitlist.enqueue(student)
        print "Course is full. " student.firstName, student.lastName, " added to waitlist."
    else
        print "Course & Waitlist is full"
    end if
end procedure
```

Frequency Count

- If statement: 1 operation
- insert operation: $\mathcal{O}(n)$
- enrolled count increment: 1 operations
- Print statement: 1 operation
- Enqueue function: 1 operation
- Print statement: 2 operations

Total count = $\mathcal{O}(n) + 6$
Time complexity = $\mathcal{O}(n)$

Algorithm 2 Insert Student into Sorted Linked List by Last Name

```
procedure INSERT(Student* newStudent)
    newNode ← new Node(newStudent)
    if head = null or newStudent.lastName < head.student.lastName then
        newNode.next ← head
        head ← newNode
    else
        current ← head
        while current.next ≠ null and current.next.student.lastName < newStudent.lastName do
            current ← current.next
        end while
        newNode.next ← current.next
        current.next ← newNode
    end if
end procedure
```

Frequency Count

- Initialization of newNode variable: 1 operation
- If statement: 1 operation
- New node assignment and head assignment: 2 operations
- While Loop comparisons: n operations
- current node assignment: n operations
- next Pointer assignment: 2 operations

Total count = $2n + 4$
Time complexity = $\mathcal{O}(n)$

Algorithm 3 Drop Student from Course

```
procedure DROPSTUDENT(string student_ID)
    if enrolledStudents.remove(student_ID) = true then
        print "Student with ID " student_ID, " has been unenrolled."
        enrolledCount ← enrolledCount −1
        if waitlist.size() > 0 then
            EnrollStudent(waitlist.front())
            waitlist.dequeue()
        end if
    else
        print "Student with ID " student_ID, " is not enrolled."
    end if
end procedure
```

Frequency Count

- If statement: 1 operation
- Remove node operation: $\mathcal{O}(n)$
- Print statement: 1 operation

- enrolled count decrement: 1 operations
- Dequeue function: 1 operation
- Print statement: 1 operations

Total count = $\mathcal{O}(n) + 5$
Time complexity = $\mathcal{O}(n)$

Algorithm 4 Remove Node by Student ID

```
function REMOVE(string student_ID)
    if head = null then
        return false
    end if
    current ← head
    while current.next ≠ null and current.next.student.studentID ≠ student_ID do
        current ← current.next
    end while
    if current.next ≠ null then
        temp ← current.next
        current.next ← current.next.next
        delete temp
        return true
    else
        return false
    end if
end function
```

Frequency Count

- If statement: 1 operation
- New node assignment: 1 operations
- While Loop comparisons: n operations
- current node assignment: n operations
- If statement: 3 operations
- Return statement: 1 operations

Total count = $2n + 6$
Time complexity = $\mathcal{O}(n)$

Algorithm 5 Display All Students in the List

```
procedure DISPLAY
    current ← head
    while current ≠ null do
        print Student in the list.
        current ← current.next
    end while
end procedure
```

Frequency Count

- New node assignment: 1 operations

- While Loop comparisons: n operations
- Print statement: n operations
- current node assignment: n operations
- Return statement: 1 operations

Total count $= 3n + 2$
Time complexity $= \mathcal{O}(n)$

Algorithm 6 Fill Courses Randomly

```
procedure FillCoursesRandomly(vector¡Course¿ courses, int userGradeLevel)
    Seed random number generator with current time
    Initialize lists of firstNames and lastNames for student details
    Initialize empty set usedIDs for unique student IDs
    Determine minFill and maxFill based on userGradeLevel:
    if userGradeLevel = 1 then
        minFill ← 16, maxFill ← 20
    else if userGradeLevel = 2 then
        minFill ← 12, maxFill ← 16
    else if userGradeLevel = 3 then
        minFill ← 8, maxFill ← 12
    else if userGradeLevel = 4 then
        minFill ← 4, maxFill ← 8
    else
        minFill ← 10, maxFill ← 15
    end if
    for each course in courses do
        randomStudentCount ← Random number between minFill and maxFill
        for each j from 1 to randomStudentCount do
            firstName ← Random element from firstNames
            lastName ← Random element from lastNames
            studentID ← GenerateUniqueID(usedIDs)
            Insert studentID into usedIDs
            gradeLevel ← Random integer between 1 and 4
            courseChoice ← course.getCourseName()
            Create newStudent with details
            course.EnrollStudent(newStudent)
        end for
        if course is full then
            waitlistCount ← Random number between 0 and 3
            for each j from 1 to waitlistCount do
                firstName ← Random element from firstNames
                lastName ← Random element from lastNames
                studentID ← GenerateUniqueID(usedIDs)
                Insert studentID into usedIDs
                studentGradeLevel ← Random integer between 1 and 4
                courseChoice ← course.getCourseName()
                Create waitlistedStudent with details
                course.AddToWaitlist(waitlistedStudent)
            end for
        end if
    end for
end procedure
```

Frequency Count

- Initializing variables: $\mathcal{O}(1)$
- First For Loop comparisons: n operations
- Second For Loop comparisons: n operations
- If statement: n operations
- Return statement: 1 operations

Total count $= n \times (n + n) + \mathcal{O}(n)$
Time complexity $= \mathcal{O}(n^2)$

4.2 Data Collection and Preprocessing

Data is collected and displayed by using the terminal of the code editor. The user is instructed to input the students first name, last name, ID, and the grade level. Then the data is stored into variables in the code. Other data for courses and students is collected through generating the random data. Student names, IDs, and grade levels are randomly generated and then stored for collection. Before user interaction, the random students generated are processed, ensuring each has a unique name and ID and then are assigned randomly to courses to simulate different environments for the user every time the program is run. User input is then validated with their name and ID through strings and the program ensures inputs and menu choices are valid. Figure 4 shows the how data is collected from the user.

5 Results

Based on the test results seen in Figures 1 through 3 of Appendix B, the program successfully simulates a course registration system for the user. The user can enter in their information and then choose from a list of options in an easy to use menu. The list of available courses can be seen with the amount of seats filled . The registration process runs correctly and the user is able to register for the course they choose. The user is also able to drop a course they're registered for and view their registered courses. The waitlist implementation was also successful as the user was added to the waitlist for a course that was full.

6 Discussion

All our functions have a time complexity of $\mathcal{O}(n)$, which is acceptable given the maximum course size of approximately 20 students. This ensures the performance remains efficient for our use case. An effective course registration process was designed. The system can streamline the process of a course registration similar to actual registration systems used by univeristies. We were also able to achieve our goal of using linked lists and queues for course registrations. The waitlist implementation using the queues was very efficient because it had a time complexity of $\mathcal{O}(1)$ and so it is scalable. Also, The user-friendly interface that was designed made the system understandable and easy to use for the user. Information tracking helped manage the user's registered courses and the program handles errors in user inputs, which ensured no bugs and ease of use for the user.

7 Conclusion

Both the front-end and the back-end of the program was entirely written in C++. This study simplified real-world scenarios by focusing on the core features of registration and waitlist management, without implementing advanced once like prioritization of students or ahndling complex course schedules. However the absence of UI and the lack of support for multiple users limits the system's applicability to real-world use cases. In the end, we were able to successfully implement the Course registration program.

8 Future Work

In the future, we could use other languages to implement the front-end and we could also add a database to the back-end to store all the students and courses. We would also want to optimize the system's data structures and performance for a larger scale by implementing techniques such as indexing or caching. Increasing the available courses and names would give more options to the user and create more randomization. Eventually we would also incorporate a server aspect for multiple users to register at the same time which would better mirror an actual registration system. Adding more details to the course list such as meeting times, professor information, and locations and expanding the system to include a user-friendly GUI would also be future improvements.

9 References

- The repository can be found at `https://github.com/krober21/Course-Registration-System`.
- Linked Lists Canvas Module: Lecture 7 and Lecture 8
- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/
- https://www.geeksforgeeks.org/c-classes-and-objects/
- https://www.geeksforgeeks.org/vector-in-cpp-stl/

A Appendix A: Code

```cpp
// Method to insert a new student into the linked list in sorted order by last name
void insert(Student* newStudent) {
    Node* newNode = new Node(newStudent); // Create a new node for the student

    // If the list is empty or the new student comes before the head
    if (!head || newStudent->lastName < head->student->lastName) {
        newNode->next = head; // Insert at the beginning
        head = newNode;       // Update head to new node
    } else {
        Node* current = head; // Start at the head
        // Find the correct position to insert the new student
        while (current->next && current->next->student->lastName < newStudent->lastName) {
            current = current->next; // Move to the next node
        }
        newNode->next = current->next; // Link new node to the next node
        current->next = newNode;       // Insert the new node
    }
}
```

Figure 1: Screenshot of code for inserting students

```cpp
bool remove(string student_ID){
    if (head == nullptr){   //Checks if the head is a nullptr
        return false;
    }
    Node* current = head;
    while(current->next != nullptr && current->next->student->studentID != student_ID){ //Searching the list for the node
        current = current->next;
    }
    // Deleting the node
    if (current->next != nullptr) {
        Node* temp = current->next; // sets current node to temp
        current->next = current->next->next;    //Changing the pointers to the next nodes
        delete temp;    // Deleting node
        return true;    //Returning true
    }
    else{
        return false;   //Return false if node is not found
    }

}
```

Figure 2: Screenshot of code for removing students

```
void display() {
    Node* current = head;
    while (current) {
        cout << current->student->firstName << " " << current->student->lastName << " (ID: "
             << current->student->studentID << ", Grade: " << current->student->gradeLevel << ")\n";
        current = current->next;
    }
}
```

Figure 3: Screenshot of code for displaying students in a class

B Appendix B: Results

```
Enter your first name:Kate
 Enter your last name:Rober
 Enter your student ID:002113513
 Enter your grade level (1: Freshman, 2: Sophomore, 3
: Junior, 4: Senior):1

1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:1

Available Courses:
-----------------------------------------------------
Index     Course Name             Seats Filled
-----------------------------------------------------
1          ENGL 1000           16/20 Seats Filled
2          ENGL 2101           18/20 Seats Filled
3          ENGL 3610           17/20 Seats Filled
4          ENGL 4695           18/20 Seats Filled
5          MATH 1000           19/20 Seats Filled
6          MATH 2331           19/20 Seats Filled
7          MATH 3081           20/20 Seats Filled
8          MATH 4270           16/20 Seats Filled
9          BIOL 1000           16/20 Seats Filled
10         BIOL 2110           16/20 Seats Filled
11         BIOL 3050           19/20 Seats Filled
12         BIOL 4823           19/20 Seats Filled
13         HIST 1000           19/20 Seats Filled
14         HIST 2071           18/20 Seats Filled
15         HIST 3631           16/20 Seats Filled
16         HIST 4120           16/20 Seats Filled
17         ECON 1000           16/20 Seats Filled
18         ECON 2440           20/20 Seats Filled
19         ECON 3811           18/20 Seats Filled
20         ECON 4921           17/20 Seats Filled
-----------------------------------------------------
```

Figure 4: Screenshot of User adding information and viewing course list

```
1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:2

Enter the name of the course you want to register in (ex: ENGL 1000):ENGL 1000
 Enrollment successful for Kate Rober.

1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:5

Courses you are registered for:
- ENGL 1000

1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:3

Enter the name of the course you want to drop:ENGL 1000
 Student with ID 002113513 has been unenrolled.
You have successfully dropped ENGL 1000.

1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:5
 You are not registered for any courses.
```

Figure 5: Screenshot of User registering for courses and dropping courses

```
1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:2

Enter the name of the course you want to register in (ex: ENGL 1000):MATH 3081
 Course is full. Kate Rober added to waitlist.
Enrollment successful for Kate Rober.

1. View Course List
2. Register for a Course
3. Drop a Course
4. View Registered Students in a Course
5. View Registered Courses
6. Exit Registration
Enter your choice:4

Enter the name of the course to view registered students (ex: ENGL 1000):MATH 3081

Registered Students in MATH 3081:

Enrolled Students:
Max Allen (ID: 002002460, Grade: 1)
Ryan Anderson (ID: 002000711, Grade: 4)
Mia Blunt (ID: 002032588, Grade: 2)
Emma Carlisle (ID: 002026754, Grade: 3)
Maggie Finns (ID: 002014367, Grade: 3)
Jane Gilligan (ID: 002013987, Grade: 3)
Liam Johnson (ID: 002000482, Grade: 4)
Anna Johnson (ID: 002024243, Grade: 4)
Rachel Jones (ID: 002029902, Grade: 2)
Mia Martin (ID: 002024253, Grade: 2)
Olivia Martinez (ID: 002031606, Grade: 4)
Liam Martinez (ID: 002010910, Grade: 1)
Ryan Martinez (ID: 002022717, Grade: 4)
Sarah Miller (ID: 002002698, Grade: 3)
Elijah Moore (ID: 002021078, Grade: 1)
Meghan Robinson (ID: 002017491, Grade: 4)
Mia Robinson (ID: 002012390, Grade: 1)
Rachel Taylor (ID: 002026588, Grade: 4)
Eric Thompson (ID: 002002705, Grade: 4)
Carlos Wright (ID: 002015749, Grade: 4)

Waitlisted Students:
Olivia Smith (ID: 002001088, Grade: 3)
Liam Blunt (ID: 002005754, Grade: 1)
Kate Rober (ID: 002113513, Grade: 1)
```

Figure 6: Screenshot of User being added to waitlist and viewing other students