Data Structures

BLG 223E

Project 1

Res. Assist. Meral Kuyucu
korkmazmer@itu.edu.tr

Instructors:
Prof. Dr. Tolga Ovatman
Asst. Prof. Dr. Yusuf Hüseyin Şahin

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 03.11.2024

1. Exam Scheduling System Challenge

1.1. Mission Briefing

Dear Computer Engineering Students of Istanbul Technical University,

As you know, we now have the privilege of our very own building. This exciting development means that all of our exams will be held in our dedicated space. With this newfound autonomy, we now have the power to schedule and conduct our exams without any external interference. But, as the wise Ben Parker once said, “With great power comes great responsibility.”

As proud members of the ITU Computer Engineering family, you have been entrusted with an important responsibility. Your mission, should you choose to accept it (and I strongly suggest you do, as it counts toward your grade), is to develop an exam scheduling system for our department. This system will help us manage the exam scheduling chaos, allowing us to conduct as many exams as possible! Exciting, right?

1.2. Blueprint of the Exam System

Now that you’ve accepted your mission, it’s time to dive into the challenge. As part of our exam scheduling system, we need to represent the days of the week in a way that reflects the continuous flow of exams. To accomplish this, you will create a structure that stores the days of the week in a circular fashion–think of it like a cycle, where after Sunday, the list loops back to Monday. The system will represent the current week as of the day you’re on. For example, once we move past Tuesday, the Monday in the list will now refer to the next Monday, continuing in a seamless weekly loop. To keep things simple, you only need to schedule exams within the current week, meaning you can’t schedule an exam more than one week in advance. Your implementation will only handle the seven days of the current week—nothing beyond that.

There should be a day in your system for each day of the week (yes, we do conduct exams on the weekends, which you probably already know). Each day in the system should be a point of access to the exams that are scheduled on that particular day (organized in chronological order). You can think of the day as the starting point for accessing all the exams happening on that day. Thus, the day should contain the information of the first exam scheduled.

Each exam will contain the following key information:

- **Start Time:** Integer [8-17].
- **End Time:** Integer [9-20].
- **Course Code:** String (Ex: BLG223E)
- The next exam.

Kindly note that exams should be linked together in a way that allows you to navigate through the entire schedule for that day, one exam after the other. The last piece of information in the exam data structure should be used for this purpose.

1.3. Your Toolkit: Operations to Master the Schedule

1.3.1. Receiving Scheduling Intel

The first task is to load an existing exam schedule from a file. This schedule is currently being managed by a very overwhelmed and slightly depressed administrator who has been dealing with far too many exam rescheduling requests. The file will contain the existing exams for each day of the week. Your system should be able to read this file, build the necessary data structures, and allow further operations. An example file has been provided with the supplementary material. You can assume that this file has no errors or conflicts. You can also assume that the input files used to test your implementation will always be valid.

1.3.2. Reporting Back to Headquarters

The second important task is to write the current state of the schedule onto a file in the same format as the one you've received from the previous administrator. It is important that this file be in the same format as the one you've received, as this is what everyone is used to reading.

1.3.3. System Control: Core Exam Schedule Operations

There are 5 core operations that you must implement to control the scheduling system.

- **Add Exam:** Add a new exam to the current schedule. Create a new exam with the specifications and add it to the proper day in chronological order. If the desired time slot is already occupied by another exam, you will need to select the next available time. This could be a later time within the same day, or an earlier time in the next day. Your system should ensure that no two exams overlap (not even partially) and that the exam is placed at the next nearest possible time to the original request. You do not need to consider duplicate exams. If a request for an exam arrives in the same day/time and location, it is considered another exam regardless of the course code.
- **Remove Exam:** Remove an exam from the schedule, possibly because the professor was called to an emergency meeting (or their cat deleted their exam paper). Ensure that the schedule remains intact by properly linking any remaining exams that follow the removed exam.
- **Update Exam:** Change the time and/or day of an exam, as sometimes professors go off to a conference and lucky students get their exam rescheduled. If the

requested reschedule is not possible due to a conflict, the system should not make any changes, and the professor will need to submit a new request.

- **Clear Day:** A massive snowstorm hit Istanbul, and no one is getting to campus that day. Enjoy the surprise holiday! However, the exams scheduled for this day still need to take place. Your task is to redistribute these exams to the remaining days of the week while maintaining chronological order within each day. The first exam from the snow day should be scheduled in the first available time slot starting from the next day. The second exam should go into the next available slot, and so on. Ensure that no scheduling conflicts occur as you reinsert these exams into the schedules of the remaining days.

- **Clear Schedule:** The semester has finally come to an end, and with it, all exams are officially over. Time to relax and enjoy your freedom–only until next semester, of course...

1.4. Engineering the System: Data Structures and Functions

Listing 1.1 outlines the data structures you'll be using to maintain the circular weekly schedule. In Listing 1.2, you'll find the function prototypes critical to managing this system. Be sure that the return values and output messages follow the provided protocol closely. These outputs will be taken into consideration during your evaluation.

You are free to reuse the required functions you've already developed for other purposes. In some cases, you might find yourself calling one required function within another. As a result, the print messages from different functions may stack up in the terminal, and that's okay. Your work will be evaluated based on the numerical outputs of the functions. The terminal outputs will be used to trace the progression of your code when grading.

You are welcome to create helper functions to keep your code clean and organized. However, for the required functions, you must follow the instructions exactly. This means using the exact function names, the correct number of inputs, and the specified output types. If you deviate from these, your code will fail the tests. So, make sure to stick to the details provided and stay sharp! Every detail matters if you want to complete this mission successfully!

```
1  struct Exam {
2      int startTime;
3      int endTime;
4      std::string courseCode;
5      Exam* next; // Next exam in the day
6
7      Exam(int sTime, int eTime, std::string code)
8          : startTime(sTime), endTime(eTime), courseCode(code), next(
               nullptr) {}
```

```
};

struct Day {
    std::string dayName;
    Day* nextDay; // Next day in the circular list
    Exam* examList; // Head of the exam list for this day

    Day(std::string name) : dayName(name), examList(nullptr), nextDay(
        nullptr) {}
};

struct Schedule {
    Day* head; // Head of the circular linked list of days

    Schedule() : head(nullptr) {}
};
```

Listing 1.1: Exam, Day, and Schedule Structures

```

Schedule* CreateSchedule();
// Creates a new schedule with 7 days, linking them in a circular list,
    with the head pointing to Monday.
//Print "Schedule creation complete."

int AddExamToSchedule(Schedule* schedule, int startTime, int endTime,
    const std::string& courseCode);
// Adds a new exam to the specified day in the schedule in chronological
     order.
// Returns 0 if added to schedule on the desired day/time.
// Print "[courseCode] exam added to [day] at time [startTime] to [
    endTime] without conflict."

// Returns 1 if added to schedule but on a later day/time.
// Print "[courseCode] exam added to [day] at time [startTime] to [
    endTime] due to conflict."

// Returns 2 if schedule is completely full.
// Print "Schedule full. Exam cannot be added."

// Returns 3 if exam duration is longer than 3 hours, or if start/end
    time falls outside of allowed hours.
// Print "Invalid exam."

int RemoveExamFromSchedule(Schedule* schedule, const std::string& day,
    int startTime);
// Removes an exam from the specified day in the schedule based on the
    start time.
// Returns 0 if exam is found and removed.
```

```
23 // Print "Exam removed successfully."
24
25 // Returns 1 if no such exam exists.
26 // Print "Exam could not be found."
27
28 int UpdateExam(Schedule* schedule, const std::string& oldDay, int
       oldStartTime, const std::string& newDay, int newStartTime, int
       newEndTime);
29 // Updates an existing exam, changing its time and/or day.
30
31 // Returns 0 if update is successful.
32 // Print "Update successful."
33
34 // Returns 1 if update cannot be made.
35 // Print "Update unsuccessful."
36
37 // Returns 2 if no such exam can be found.
38 // Print "Exam could not be found."
39
40 // Returns 3 if new exam duration is longer than 3 hours or if start/
       end time falls outside of allowed hours.
41 // Print "Invalid exam."
42
43 int ClearDay(Schedule* schedule, const std::string& day);
44 // Clears all exams from the specified day in the schedule.
45 // Returns 0 if the day had exams, and all exams were relocated
       successfully.
46 // Print "[Day] is cleared, exams relocated."
47
48 // Returns 1 if the day was already empty when the function was called.
49 // Print "[Day] is already clear."
50
51 // Returns 2 if the day had exams, but the schedule is full, and the
       exams could not be relocated.
52 // Print "Schedule full. Exams from [Day] could not be relocated."
53
54 void DeleteSchedule(Schedule* schedule);
55 // Destructs the entire schedule by removing all exams from all days and
        deallocating memory.
56 // Print "Schedule is cleared."
```

Listing 1.2: Function Prototypes

2. Sample Run

2.1. Operations and Mission Results

Let’s simulate a mission scenario by walking through a sample input file. After receiving the initial schedule (the "mission intel"), we’ll demonstrate a series of operations that manipulate the schedule. You will witness how the system adapts to new tasks, performs the required operations, and produces the final output file (the "mission report"). This walk through will provide a clear example of how your system should function, allowing you to verify your own results and ensure your mission is a success.

Sample Input

File Name: `ExamScheduleInput.txt`

```
Monday
9 10 BLG113E
11 12 BLG212E
14 16 BLG231E

Tuesday
10 11 BLG223E
13 14 BLG435E

Wednesday
8 10 BLG317E
12 14 BLG335E

Thursday
9 10 BLG433E
11 13 BLG351E

Friday
8 10 BLG411E
10 12 BLG336E

Saturday
9 11 BLG348E

Sunday
10 12 BLG439E
```

Scheduling Operations

```
1. AddExamToSchedule(schedule, "Tuesday", 15, 17, "BLG411E")
    -> BLG411E exam added to Tuesday at time 15 to 17 without conflict.
    Return Value: 0

2. AddExamToSchedule(schedule, "Monday", 16, 18, "BLG439E")
    -> BLG439E exam added to Monday at time 16 to 18 due to conflict.
    Return Value: 1

3. AddExamToSchedule(schedule, "Sunday", 20, 23, "BLG439E")
    -> Invalid exam.
    Return Value: 3

4. RemoveExamFromSchedule(schedule, "Thursday", 11)
    -> Exam removed successfully.
    Return Value: 0

5. RemoveExamFromSchedule(schedule, "Tuesday", 9)
    -> Exam could not be found.
    Return Value: 1

6. UpdateExam(schedule, "Monday", 9, "Monday", 10, 11)
    -> Update successful.
    Return Value: 0

7. UpdateExam(schedule, "Tuesday", 10, "Wednesday", 12, 13)
    -> Update unsuccessful.
    Return Value: 1

8. UpdateExam(schedule, "Wednesday", 15, "Wednesday", 12, 14)
    -> Exam could not be found.
    Return Value: 2

9. UpdateExam(schedule, "Sunday", 10, "Sunday", 20, 23)
    -> Invalid exam.
    Return Value: 3

10. ClearDay(schedule, "Wednesday")
    -> Wednesday is cleared.
    Return Value: 0

11. ClearDay(schedule, "Wednesday")
    -> Wednesday is already clear.
    Return Value: 1
```

Sample Output File

File Name: `ExamScheduleOutput.txt`

```
Monday
10 11 BLG113E
11 12 BLG212E
14 16 BLG231E
16 18 BLG439E

Tuesday
10 11 BLG223E
13 14 BLG435E
15 17 BLG411E

Wednesday
(No exams scheduled)

Thursday
9 10 BLG433E
10 12 BLG317E
12 14 BLG335E

Friday
8 10 BLG411E
10 12 BLG336E

Saturday
9 11 BLG348E

Sunday
10 12 BLG439E
```

3. Deliverables for Successful Completion

You will be provided with the following files: **include/exam.h, include/schedule.h, src/exam.c, src/schedule.c and src/main.c**, and the necessary test files. Your task is to complete the implementation by filling in the provided header and source files. As you work, you should ensure that your solution runs correctly by using the test files to validate your progress.

Feel free to modify main.c to view intermediate results as you work through the problem. However, keep in mind that while you may adjust main.c, I will be testing your solution with a different version of the file.

Also note that the provided test files will not be the only ones used for grading. Your code will be evaluated on additional test cases, so make sure your implementation is robust and can handle various scenarios.

IMPORTANT NOTE:

- Copying code fragments from any source, including books, websites, or classmates, is considered plagiarism.
- You are free to conduct research on the topics of the assignment to gain a better understanding of the concepts at an algorithmic level.
- Refrain from posting your code or report on any public platform (e.g., GitHub) before the deadline of the assignment.
- You are **NOT** permitted to use the STL for any purposes in this assignment.
- Additionally, please refrain from using any AI tools to help you complete your work. While I cannot directly detect AI-generated code, relying on such tools is not in your best interest. Everything you learn in this class forms the foundation for future courses, and if you do not build a strong foundation now, you will likely struggle in the semesters to come.