# Project 2: Event Scheduler

## 1. Introduction

In this assignment, you will implement two modules, Event and Scheduler (+ Node), to represent event schedulers in the program. The code skeleton provides you with four files (**Event.hpp**, **Scheduler.hpp**, **Node.hpp**, **test.cpp**) and is under **/comp/15/files/p2** on the CS server. Your jobs are to implement Event, Scheduler, and Node classes, as well as to test the functionalities of your modules by writing programs that use your modules.

1. Log in to the CS homework server.
2. Move to your **comp15** directory that you created in Lab 1.
3. Create a directory named **project2** under the comp15 directory.
4. Move to the project2 directory.
5. Copy the code skeleton to the current working directory.
6. Leverage the features of Git to manage your progress toward writing the program.

As the first step, you are encouraged to take a look at the contents of the code skeleton and try identifying what each file is for. Then, spend enough time to think of how you represent event schedulers in your program. As a part of the program specifications, you are asked to represent an event scheduler as a singly linked list of events in your program.

**Suggestion:** Consider to make the scheduler so its events are ordered by their start times.

You will at least need to create **Event.cpp**, **Scheduler.cpp** and **Node.cpp** files by yourself. Note that the file names matter for the grading purpose and are case-sensitive. Note also that the public section of each **Event.hpp** and **Scheduler.hpp** as well as the entire **Node.hpp** cannot be modified. Finally, you are expected to write your own tests in **test.cpp**.

## 2. Specifications

### Part 1: Event class

`Event();`

This event is initialized to be one that holds (day) **1**, (hour) **0**, (minute) **0**, and (description (case-sensitive)): **SAMPLE**

`Event(int day, int hour, int minute, std::string description);`

This event is initialized to hold the given data. Note: The given day, hour, and minute are for representing the start time of this event. You can assume the input is valid.

`Event(const Event& other);`
`Event& operator=(const Event& other);`

This event is initialized to be one that holds the same information that the given event holds.

`~Event();`

You are supposed to release (memory) resources, if necessary.

`bool startBefore(const Event* another) const;`

Return true if the start time of this event is any time before the start time of the given event; false, otherwise. If the "another" points to `nullptr`, return false.

`bool startAfter(const Event* another) const;`

Return true if the start time of this event is any time after the start time of the given event; false, otherwise. If the "another" points to `nullptr`, return false.

`bool startAtSameTime(const Event* another) const;`

Return true if the start time of this event is the same as the start time of the given event; false, otherwise. If the "another" points to `nullptr`, return false.

`bool startBefore(int day, int hour, int minute) const;`

Return true if the start time of this event is any time before the given time; false, otherwise.

`bool startAfter(int day, int hour, int minute) const;`

Return true if the start time of this event is any time after the given time; false, otherwise.

`bool startAt(int day, int hour, int minute) const;`

Return true if the start time of this event is the same as the given time; false, otherwise.

`int getDay() const;`

Return the day information that this event holds.

`int getHour() const;`

Return the hour information that this event holds.

`int getMinute() const;`

Return the minute information that this event holds.

`std::string getDescription() const;`

Return the description (case-sensitive) that this event holds.

`std::string toString() const;`

Return the string representation of this event (case-sensitive). The representation should have the form **[dd HH:MM] DESCRIPTION** where:

- It begins with a open square bracket,
- followed by **dd** that is the two-digit representation of the day of this event,
- followed by a space,
- followed by **HH** that is the two-digit representation of the hour of this event,
- followed by a colon,
- followed by **MM** that is the two-digit representation of the minute of this event,
- followed by a close square bracket,
- followed by a space,
- followed by **DESCRIPTION** that is the description of this event.

For example, the string representation of an event holding (day) **1**, (hour) **0**, (minute) **30**, and (description) **go to bed**: **[01 00:30] go to bed**
For example, the string representation of an event holding (day) **21**, (hour) **19**, (minute) **5**, and (description) **dinner**: **[21 19:05] dinner**

## Part 2: Scheduler class

`Scheduler();`

This scheduler is initialized to be one is for (year) **2019** and (month) **6**, i.e. June.

`Scheduler(int year, int month);`

This scheduler is initialized to be one for the given year and month.
If the given year is not a positive integer and/or if the given month is not in the range between 1 and 12 (inclusive), then throw a `std::range_error` exception with the message (case-sensitive): **Out of Range**

`Scheduler(const Scheduler& other);`
`Scheduler& operator=(const Scheduler& other);`

This scheduler is initialized to be one that holds the same information (deep copy) that the given scheduler holds.

`~Scheduler();`

You are supposed to release (memory) resources used for this scheduler.

`void add(Event* event);`

Add the given event to this scheduler.
If the "event" points to `nullptr`, then do nothing.
If the given event starts at the same time that any event, which this scheduler holds, does, then throw a `std::runtime_error` exception with the message (case-sensitive): **Event Time Conflict**

If the given event holds information that does not meet the requirements listed below, then throw a `std::runtime_error` exception with the message (case-sensitive):
**Not Legal Event**

Requirements:

- The day of the given event must be in the range based on the year and month that this scheduler holds. e.g. We have 30 days in June 2019 whereas we have 31 days in July 2019.
- The hour of the given event must be between 0 and 23 (inclusive).
- The minute of the given event must be between 0 and 59 (inclusive).
- The description of the given event must be at least one character long.

`std::string getFirstEventAfter(int day, int hour, int minute) const;`

Return the string representation of the first event, which this scheduler holds, that starts at or any time after the given time.
If no such event is found, then return a string (case-sensitive): **No Event Found**

`void removeAllEventsOn(int day);`

Remove all events, which this scheduler holds, that start on the given day.

`void removeAllEvents();`

Remove all events that this scheduler holds.

`int getNumberOfEventsOn(int day) const;`

Return the number of events, which this scheduler holds, that start on the given day.

`int getNumberOfEvents() const;`

Return the number of events that this scheduler holds.

`bool isEmpty() const;`

Return true if this scheduler holds no events; false, otherwise.

`int getYear() const;`

Return the year that this scheduler holds.

`int getMonth() const;`

Return the month that this scheduler holds.

`std::string toString() const;`

Return the string representation of this scheduler (case-sensitive). The representation should have the form such that:

- The first line should have the form: **yyyy MMM.**

    - where **yyyy** is the four-digit representation of the year that this scheduler holds and **MMM** is the three-character representation of the month that this scheduler holds.
- followed by a period
- followed by a newline control: **\n**

- The first line is followed by lines each of which is the string representation of an event that this scheduler holds, followed by the newline control, except for the last event.
- Note that events are ordered by their start times, so earlier event appears first in the string representation.

For example, when the string representation of a scheduler that holds (year) **2019**, (month) **6** and two events (day, hour, minute, description), {**1**, **0**, **0**, **SAMPLE**} and {**20**, **18**, **30**, **TASK**} is printed, it should look as following:
**2019 Jun.**
**[01 00:00] SAMPLE**
**[20 18:30] TASK**

(Note: There is no newline control after the **TASK**.)

## Part 3: Node class

`Node();`
`Node(const Node& other);`
`Node& operator=(const Node& other);`

Throw a `std::runtime_error` exception with the message: **Not Implemented**

`Node(Event* data);`

This node is initialized to be one that holds the given data. Note: Be aware of the type of the argument.

`~Node();`

You are supposed to release (memory) resources. Note: Be aware that the Scheduler class also has its own destructor.

`Event* getData() const;`

Return the data that this node holds. Note: Be aware of the return type.

`Node* getNext() const;`

Return the next node of this node. Note: Be aware of the return type.

`void setNext(Node* next);`

Set the given node to be the next node of this node. Note: Be aware of the type of the argument.

### Another requirement

You are responsible for your modules not producing any memory leaks and memory errors.

## 3. Testing

Modify the **test.cpp** by adding your own test cases to ensure the functionalities of your modules. The given **test.cpp** includes one example test case using assert(). You are encouraged to add as many test cases as necessary to make you feel confident about your modules.

## 4. README

Create the README file that includes the following categories with appropriate section headers.

1. **Name**: The programmer's name.
2. **Date**: The last updated date of the program code, including README.
3. **Summary**: A brief summary of the project. (e.g. What the program does? How to use the modules? and so on.)
4. **Files**: A list of files that are necessary to build and test the program.
5. **Instructions**: A sequence of commands to compile and test the program. Note that you are expected to report procedures without using the make command.
6. **References**: A list of citations to information used to complete the project.
7. **Post evaluation on planning**: Did your project go smooth, as planned? If not, which parts? What were the causes? How will you improve your planning for future assignments?

## 5. Submission

Submit your files listed below using Gradescope.
Files: **Event.hpp Event.cpp Scheduler.hpp Scheduler.cpp Node.hpp Node.cpp test.cpp README**