Project 1

COP 4530, Spring 2020

Due February 10, 2020

1 Overview

In this project, you will develop code for a scheduling application. Specifically, you will need to implement two classes, Scheduler and ScheduleNode. Scheduler is the "main" class, and it maintains the schedule as a doubly-linked collection of ScheduleNodes. Each ScheduleNode has a name and duration and represents a single event on the schedule. A header for both classes has been provided, and you are not allowed to modify this file.

The program starts with an empty schedule of length 100, to which the user may add (and later cancel) events. As user adds and removes events to the schedule, there are five main challenges your code should overcome:

- All valid changes (e.g., scheduling a new event during a free period or cancelling an existing event) should be reflected when the schedule is printed.
- Scheduling or cancelling events should not change the total length of the schedule.
- The schedule should never show two consecutive events (or free periods) with the same name. Instead, consecutive events with the same name should be merged together into a single event with a duration equal to the sum of the original events.
- Every event must have a positive (i.e., non-zero) duration.
- All allocated ScheduleNodes must be deallocated.

2 Member function descriptions

The member functions you will need to implement are as follows:

- `bool Scheduler::isAvailable(unsigned start, unsigned duration) const`: returns whether the time period from time `start` to `start + duration` is free; times are considered free if they occur during a ScheduleNode with `name == "FREE"`. Valid times are 0–99; the `head` SchedulerNode is considered to start at time 0, and the `next` node starts immediately after the previous node ends. Times beyond the Scheduler's `length` are not considered free.
- `void Scheduler::schedule(const string& name, unsigned start, unsigned duration)`: schedule a new event with the given name and duration at a given time, making sure not to create an invalid schedule (e.g., length $\neq 100$, consecutive nodes with same name, or 0-length node). You may assume that the user has already called `isAvailable` on the start and duration.

- `void Scheduler::free(unsigned start)`: frees up the time associated with the event that starts at the given time (i.e., sets that time back to `FREE`), without creating an invalid schedule. Has no effect if the time is already free or no event starts at that time.
- `void Scheduler::printSchedule() const`: prints information for all of the events in the schedule, in the following format:

  `TIME:  NAME (DURATION)`

  where `TIME` represents the start time of the event, `NAME` the event's name, and `DURATION` the length of the event. Note that there are two spaces after the colon and one before the open parenthesis.
- `Scheduler::~Scheduler()`: destroys the Scheduler, freeing all ScheduleNodes. You are not required to implement a copy constructor or copy assignment operator for the Scheduler.
- `void ScheduleNode::merge(ScheduleNode* other)`: *(Optional)* compares this ScheduleNode to its argument (which should be the next or previous node), and combines the two nodes into one with the same name where the length is the sum of the two original nodes. This function is not called by the driver and is totally optional, but you may find it helpful to implement this so that you can use it in other functions.

3 Driver file

You have been provided with a simple driver file that parses commands from `cin` and calls relevant `Scheduler` functions. The first command it recognizes is `print`, which just calls `printSchedule`.

The second command is `add`, which will add events to the schedule with `schedule`. The format for this command is:

```
add EVENT from TIME1 to TIME2
```

where `EVENT` should be replaced by the name of the event, `TIME1` with the start time, and `TIME2` with the ending time. If the indicated time is not available (`!isAvailable`), the driver will print an error message (`Scheduling conflict:  not added`), or it will print a notification that the change was successful (`Added to schedule`).

The third command is `cancel`, which will remove events from the schedule with `free`. The format for this command is:

```
cancel TIME
```

where `TIME` should be replaced with the start time of the event to cancel. Both the `add` and `cancel` will print the modified schedule afterwards.

The last command recognized by the driver is `quit`, which ends the program. The driver can also be exited with `Ctrl-C`. All commands (and event names) are case-insensitive.

4 Submission and grading

You should submit only your source code, in `scheduler.cpp`. Your code will be evaluated based on whether it produces the correct output for some number of test cases. You may use any development

environment that you like when developing the code; however, it will be compiled and run using `g++` in a Linux environment. Code that does not compile will not receive substantial credit, so be sure that your code can be compiled using `g++`.

The development environment used in class is VS Code (`https://code.visualstudio.com/download`). This IDE includes useful features like syntax highlighting and an integrated terminal window that you can use to invoke a compiler.

Windows users can download MinGW (`https://sourceforge.net/projects/mingw-w64`) to use `g++`, though you will need to add the MinGW bin directory to the Windows path in order for the terminal to recognize the command `g++` (or `gdb` or any of the other tools provided by MinGW).