Istanbul Technical University
Computer Engineering Department
BLG 233E Data Structures, Fall 2019
Homework #2
Due November 13, 2019 11:59pm

Problem Defintion

Doctor Celebi is a psychologist who works in his own office. He stores patient appointments in a text file. He just stores the initial characters of name and surnames of patients. He usually needs to take unscheduled travels, so he gives a number for appointment day e.g. 5, 23 or 35 instead of a date. When he does not have a trip, he comes to work at 8:00 o'clock in the morning. And he leaves the office after the last appointment of the day. Each appointment takes one hour. His assistant arranges the appointments by day and time manually which is error-prone and time consuming. Therefore, these days, they are looking for a programmer to write an app that does this arrangement process automatically. Adding task ascending order with respect to day number and time, removing task, delaying one task to the next time or delaying all tasks of a day will be performed by this app. Some important instructions about this app as below;

- A part from the text file noted is given below as example(Figure 1).Each row is an appointment information for a patient.

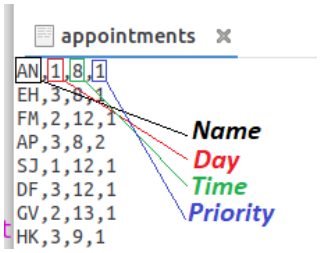

Figure 1: The text file that contains appointments info

- Each task (Task will be used instead of appointment in this document and vice verse.) has a name (two capital letters e.g. AN: Ali Nevehirli), a day number (an integer bigger than 0), an hour (an integer between 0 and 23 , 24 hour style) and priority degree (an integer between 1 and 3) info. 3 is the top priority and 1 is lowest priority. (Do not care about minute, we are just dealing with hour)
- The appointments will be arranged by day number and time in ascending order. The first task of each day points out the first tasks of the next and previous days (Figure 2). Also, each task has a pointer that keeps the next task in the same day. Cycled list allows you to use shortest path to achieve target place in operations such as adding, removing etc.
- If there are more than one task in the same day and time, take into account the priorities of tasks. The task which has smaller priority is delayed to **first next available time**. Available times are explained in the following paragraph. If available time does not exist, the task is delayed to next free (not allocated) hour between 8:00 and 16:00(in the current day or following day).
- **Available times** are the times when Dr Celebi is at office and an appointment does not exist. The available times are restricted between 8:00 and 16:00. After 16:00 o'clock there is not any available time even if Dr Celebi is at office.Do not forget that **Dr Celebi leaves from office after the last appointment in a day.** The next available times for an appointment are the available times that comes after the appointment. An example of appointment schedule (Figure 3) and next available times according to this schedule (Figure 4, Figure 5) are given below;

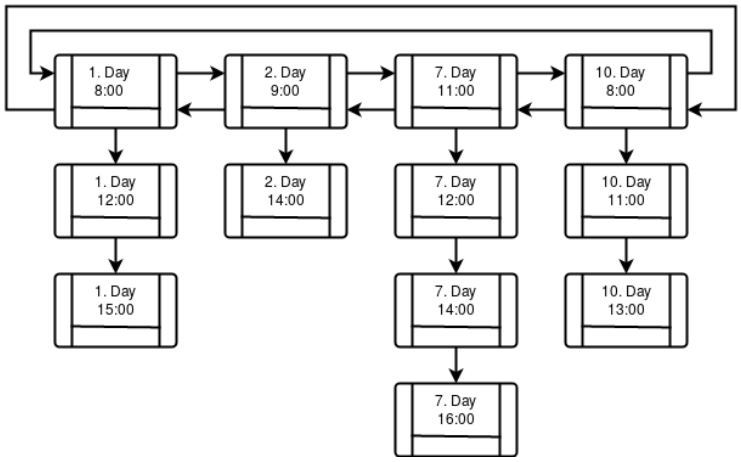

Figure 2: The data structure, circular multi-linked list that is required to use in this implementation

```
1. DAY
         8:00   RD(2)
         9:00   RF(1)
        10:00   AN(1)
        12:00   SJ(1)
2. DAY
        12:00   FM(1)
        13:00   GA(2)
        14:00   GV(1)
        15:00   VN(1)
3. DAY
         8:00   AP(2)
         9:00   EH(1)
        10:00   HK(1)
        11:00   ZE(1)
        12:00   DF(1)
4. DAY
         8:00   YS(1)
         9:00   HD(1)
        11:00   CA(2)
        12:00   CS(1)
        15:00   MC(1)
5. DAY
        15:00   KB(1)
10. DAY
         9:00   HM(1)
        15:00   RM(1)
```

Figure 3: An example schedule of appointments

Implementation Details

1. First of all, you are not allowed to include any Standard Template Library (**STL) container**. You will use the below data structure named as `Task`.

```
struct Task{
        char *name;
        int day;
        int time;
        int priority;

        Task *previous;
        Task *next;
        Task *counterpart;//to link the other tasks on the same day
};
```

```
Next available times for the task that is on 1.day at 10:00
1. DAY
        11:00
2. DAY
         8:00
         9:00
        10:00
        11:00
3. DAY
4. DAY
        10:00
        13:00
        14:00
5. DAY
         8:00
         9:00
        10:00
        11:00
        12:00
        13:00
        14:00
10. DAY
         8:00
        10:00
        11:00
        12:00
        13:00
        14:00
1.day and 11:00 is the first available day and hour to suspend for AN on 1.day at 10:00
```

Figure 4: Next available times for the task that is on 1st day at 10:00

```
Next available times for the task that is on 4.day at  8:00
4. DAY
        10:00
        13:00
        14:00
5. DAY
         8:00
         9:00
        10:00
        11:00
        12:00
        13:00
        14:00
10. DAY
         8:00
        10:00
        11:00
        12:00
        13:00
        14:00
4.day and 10:00 is the first available day and hour to suspend for YS on 4.day at  8:00
```

Figure 5: Next available times for the task that is on 4th day at 8:00

2. The header file of the source code that you will implement is given below. You can add new functions or variables if you need, but you are not allowed drop any of these functions or variables that is declared in given header file. You can find the source codes in the folder that is given you for this implementation.

```
#ifndef TASK_MANAGEMENT_TOOL
#define TASK_MANAGEMENT_TOOL

#include <stdio.h>
#include "task.h"

struct WorkPlan{
        void create();
        void close();
        void add(Task *task);
        void checkAvailableNextTimesFor(Task *delayed);
        int  getUsableDay();
        int getUsableTime();
        void remove(Task *target);
        void display(bool verbose, bool testing);
        void delayAllTasksOfDay(int day);
        Task * getTask(int day, int time);

        Task *head;
        int usable_day;
        int usable_time;
};
#endif
```

3. The following functions are already written. You can find them in the file named as **'task_management_tool.cpp'** Please, do not change these functions. They are explained briefly as below.

   `void display(bool verbose, bool testing)` function takes two boolean arguments that controls what will be the written to command line.

   `int getUsableDay()` function returns the `int usable_day` variable for a task.

   `int getUsableTime()` function returns the `int usable_time` variable for a task. (Please look 'Problem Definition' section for available day and time.)

4. The variables are explained as below.

   `Task *head` keeps the address of header.

   `int usable_day` and `int usable_time` keep the first available day and time respectively for a task. If the They change for each task. According to task they need to be declared again.

5. The following functions will be implemented by you. Please, do not change function declarations.

   `void create()` function will be used for initialization of the structure.

   `void close()` function will be used for termination of the structure.

`void add(Task *task)` function will add the `Task *task` to the structure according to the mentioned rules above.

`void checkAvailableNextTimesFor(Task *delayed)` function will check the available next times for a task that is given as parameter and assigns the first available day and time to the `int usable_day` and `int usable_time` variables respectively. If available time does not exist, `int usable_day` and `int usable_time` variables stores respectively the day number and time of the next free (not allocated) hour between 8:00 and 16:00 (in the current day or following day).

`void delayAllTasksOfDay(int day)` function will delay all tasks of the given day to the next available times **respecting their time order**. If available time does not exist, the tasks are delayed to the next free (not allocated) hours between 8:00 and 16:00 in the following day **respecting their time order**.

`void remove(Task *target)` function will remove the task that is given as parameter.

`Task * getTask(int day, int time)` function will return the address of the task that is on given day and time.

Given Files, Compilation and Screen Output

– Five files are given for you to do this implementation. They are briefly described as below;

You have the `app.cpp` source file to check and test your implementations.
PLEASE, DO NOT CHANGE `app.cpp` FILE.

You have a text file named as `appointments`. This file contains the appointments data that you will arrange as mentioned above.

You have `task.h` and `task_management_tool.h` header files.
PLEASE, DO NOT CHANGE `task.h` FILE.
PLEASE, DO NOT REMOVE ANY FUNCTION OR VARIABLE IN `task_management_tool.h` FILE, IF YOU NEED YOU CAN ADD NEW FUNCTIONS.

You have `task_management_tool.cpp` source file to implement this application.
PLEASE, DO NOT CHANGE ALREADY WRITTEN FUNCTIONS. YOU HAVE TO WRITE THE REQUIRED FUNCTIONS THAT IS MENTIONED ABOVE. YOU CAN ADD NEW FUNCTIONS IF YOU NEED.

– You should compile and run your implementation on Linux environment using **g++ (version 4.8.5 or later)**. For this purpose, you can use ITUs Linux Server using SSH protocol. Compile and link source files using the following commands. Finally, run executable file as below;

```
//Compiling-only
g++ -c -Wall -Werror task_management_tool.cpp -o task_management_tool.o
g++ -c -Wall -Werror app.cpp -o app.o
//Linking
g++ task_management_tool.o app.o -o app
//Running
./app
```

– Only command line outputs of `void display(bool verbose, bool testing)` function will be checked in evaluation. So, do not worry about the other functions command line outputs.

Your program will be checked using **Calico(https://bitbucket.org/uyar/calico)** automatic checker. Therefore, **make sure that the application works as desired.** You will get zero marks if the automatic checker fails.

Submission Rules

- You will summit only four files, two source and two header files, that are mentioned above. Please, do not change file names or extensions.
- Only electronic submissions through Ninova will be accepted no longer than November, 13 at 11:59pm.
- Use comments wherever necessary in your code to explain what you did.
- You may discuss the problems at an abstract level with your classmates, but you should not **share or copy code** from your classmates or from the Internet. You should submit your **own, individual** homework.
- Academic dishonesty, including cheating, plagiarism, and direct copying, is unacceptable.
- Note that **YOUR CODES WILL BE CHECKED WITH THE PLAGIARISM TOOLS!**
- Make sure you write your name and number in all of the source files of your project, in the following format:

  /* @Author
  Student Name: <student_name>
  Student ID : <student_id>
  Date: <date> */

- If there is unclear statement about this homework, you can ask your questions under the thread that is specially started for homework 2 (Questions about HW2) on the message board for BLG 233E on NINOVA. Please check before writing your question whether your question is asked by someone else. .

  Do not share any code or text that can be submitted as a part of an assignment (discussing ideas is okay).