Object Oriented Programming A, B
FAST-NU, Lahore, Spring 2019

Homework 1

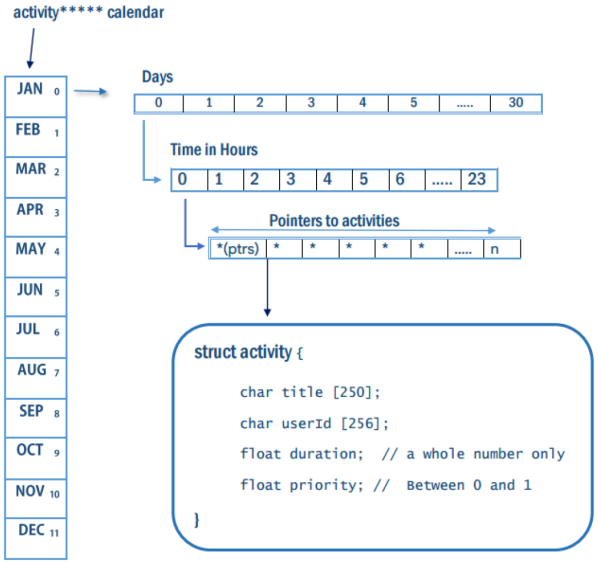
Activity Calendar & Scheduler

Due Friday February 8 11:55 PM 100 pts.

The objective of this homework is to familiarize the students with objects using structs, pointers to objects, dynamic memory allocation of objects, and the power of multi-dimensional arrays in providing direct access.

1. **What is this application about?** It is a software (console based) which stores information for all the months, days, and hourly time slots of the current year (2019), and for each slot stores activities added by a calendar administrator, where each activity comprises of: title, priority, userid, and duration. The userid is needed since there could be multiple users of the calendar application all of whose data is stored in the same calendar (think of this as a shared, network calendar). **Importantly**: activities only begin and end on whole-hours (example, 12AM, or 1PM, but not at 12:30, 01:15 etc.). However, activities can range for more than one hour, i.e. for two hours, three hours etc, so we also need to store their duration. Note that the userid is unique for each user.
   Given this information, we will be able to ask the program some interesting questions.

2. **What will be the structure of the calendar?** It is shown in the following picture:



**NOTE: Our struct activity will be slightly different from the picture, as described below:**

```
struct activity{
    char * title;//to be allocated dynamically
    char * userid;//to be allocated dynamically
    int duration;
    float priority; //between 0 & 1
};
```

- As shown in the picture, the calendar is an array of size 12 (one entry per month), where each entry is a pointer to a dynamic array whose size is equal to the number of days in that month in the year 2019.For example, January's array should have 31 entries, Feb's array should have 28 etc.
- Each entry in a month's array contains a pointer pointing to an array of the days in that month. All these day arrays are of size 24 as there are 24 hours in a day. If there's no activity in a particular day – for example, if there is no activity on January 5, then the pointer of that day's array is set to nullptr.
- Each entry in the 24-size 'days array' contains a pointer to an array of pointers to activities in that hour of that day. If an hour has no activity this pointer is nullptr. Of course, the sizes of this activities' arrays will be different, depending on the number of activities in that particular hour of that day.
- Each single activity (a box allocated on the heap) contains the information as described in the description of the activity struct above. Note that in each activity, the title and userid are dynamically allocated as well.
- So the calendar as a whole will be pointed to by a pointer of the type **activity ***** calendar;** to be allocated dynamically according these specifications.

3. **How will the data be loaded into the calendar and saved back?** Your program will be given an input file called "calendar.dat" (two sample files are provided to you with this assignment, we will give a different file at the time of evaluation), it will contain an activity and its date and time in a single line. A few example lines of the file are given below:

```
09/10,11,14,user1,act33,Mow the lawn,0.12
02/21,10,13,user1,act23,Play cricket,0.22
09/10,14,18,user3,act144,Study for exam,0.90
…
```

These lines follow the following format:
day/month,start_time,end_time,userid,actid,title,priority

**Notes on the data:**

- Items are separated by commas (without spaces)
- Start and end times are integers and range from 0 to 23 (where 0 is midnight).
- Priority is always a float between 0 and 1, where higher number means higher priority.
- Duration of an activity can be computed from start and end times.
- The entry of an activity is made at its starting hour in the calendar.
- You should ignore the actId entry (for activity id) from each line, i.e. do not load it into the calendar.

  Your program should be able to read the file and load the calendar in the format described in point 2. Also, the program should save the calendar back in the exact same format.

4. **What are the operations the calendar should be able to perform?** The user of your software is like the administrator of a database, who has access to the data of all other users. This 'admin' can ask the program a range of questions. The questions that your program should be able to answer are listed below. Each of these should translate into one or more C++ functions. For each of these operations, there should be an option available on a menu on the console.

   i. **List all activities of a given user during a time period:**
   **Input:** userid, start date (day and month), end date
   **Output:** line by line listing of activities

   ii. **List the 5 most important activities of a given user during a time period:** (importance is based on priority, high priority means high importance)
   **Input:** userid, start date, end date
   **Output:** line by line listing of activities

   iii. **For a given user, print the longest free period; that is, the longest consecutive number of days in which the user has no activity.** (How useful this feature would be while planning a vacation!)
   **Input:** userid
   **Output:** start date, end date (of the free period), number of days in this period

   iv. **List all the clashing activities of a pair of users, during a time period. (Clashing activities are those that share one hour of time or more).**
   **Input:** user id1, user id2, start date, end date
   **Output:** line by line listing of clashing activities

   v. **For a list of users, and a time period, list all hourly slots that are free for all these user in this time period.** (How useful this feature would be to plan a meeting!)
   **Input:** array of user ids, start date, end date

**Output:** line by line listing of time slots that are free for all

vi. **Print activity stats for a given month**
**Input:** month number (between 1 and 12)
**Output:** total number of activities in the month, average number of activities per day, the busiest day of the month, number of activities in the busiest day, the day with the highest average priority of the activities (i.e. the most important day), the number of activities in the most important day.

vii. **Print Calendar Stats (for the whole year)**
**Output:** total number of activities in the year, average number of activities per month, the busiest month of the year, number of activities in the busiest month

viii. **Remove a user from the calendar (remove all activities of this user)**
**Input**: userid
All the dynamically allocated memory associated for this user must be deallocated as well.

ix. **Save the calendar**
Save the calendar back into the file.

x. **Print a month of the calendar**
Input: **a month number**
**Output:** All the day numbers of that month printed on the screen, with all the days with some activities printed in white, and all the days with no activities printed in yellow. You should print this month as a block of number, just as it is printed in a standard wall calendar.

**At the end of the program all the dynamically allocated memory must be deallocated**.

**THE END**