

**Project 2**

**Activity Calendar and Scheduler**

|  |  |
| --- | --- |
| **Release Date** | Thursday, March 12, 2015, 11:55PM |
|  |  |
| **Due Date** | Thursday, March 26, 2015, Before Class, on SLATE |
|  |  |
| **Absolute Weight** | 5% |
|  |  |

**Statement**

The objective of this programming project is to familiarize the students with pointers and dynamic memory allocation in multiple dimensions. In the following sections I shall describe the structure of the program, I/O, and required functionalities.

1. **About the Application**

It is a software (console based) which stores all the months, days, and hourly time slots of the current year (2015), and for each slot stores activities added by a calendar user, where each activity comprises of: title, priority, user-id, activity-id, and duration. The user-id is needed since there could be multiple users of the calendar application. 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 need to store their duration. Note that the user-id is unique for each user and activity-id is unique for each activity. Given this information, we can ask the program some interesting questions.

1. **Structure of the Calendar**
   * The calendar is an array of size 12 (one entry per month), where each entry is a pointer to a dynamic array of size equal to the number of days in that month in the year 2015.
   * Each entry in a month array (which represents a day) is a pointer to an array of size 24, for the 24 hours in that day (this pointer is null if the day has no activity).
   * Each entry in the day array (which represents an hour) is a pointer to an array of activities beginning at that hour. If an hour has no activity beginning from it, this pointer is null. Of course, the size of an activity array is equal to the number of activities in that day. **Important note:** you should mark the end of each activity array with an activity with empty title "", so that when you are traversing the array you will know where to stop.
   * Each activity is a struct Activity type object, containing the following members:   
     title, user\_id, act\_id, priority, duration  
     for example, these members may receive the values: "Pay the cook", "user99", "act2001", 0.75 and 2 respectively.  
     **Programming Tip 1:** Write a function void parseActivity(string act, Activity & a) [or void parseActivity(char act[], Activity & a), if you like to work with character arrays directly]. It takes the activity string 'act' as input (this comma separated string comes from a file: it is one line of the file), tokenizes it on "," and stores the separated values in the Activity object 'a' passed by reference.   
     **Programming Tip 2:** To separate out the components in a string ( a process called tokenization) you can use one of the several methods available in C++. One way is to use the [strtok](http://www.cplusplus.com/reference/cstring/strtok/) function on character arrays. Another method is to use the built-in functions of the [string](http://www.cplusplus.com/reference/string/string/) type. In particular, look into the functions [find\_first\_of](http://www.cplusplus.com/reference/string/string/find_first_of/) and [substr](http://www.cplusplus.com/reference/string/string/substr/). Use the method that suits you best, though I recommend the latter.  
     **Programming Tip 3:** You might find these functions useful: [atoi](http://www.cplusplus.com/reference/cstdlib/atoi/), [itoa](http://www.cplusplus.com/reference/cstdlib/itoa/), [atof](http://www.cplusplus.com/reference/cstdlib/atof/).
   * The calendar then is a four dimensional pointer: Activity \*\*\*\* calendar; to be allocated dynamically according to the descriptions above.
2. **Loading and Saving**

Your program will be given an input file (two sample files have been provided with this assignment, but 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 are given below:  
10/09,11,14,user1,act33,Mow the lawn,0.12  
21/2,10,13,user1,act23,Play cricket,0.22  
11/09,14,18,user3,act144,Study for exam,0.90  
...  
These lines follow the following format:  
*day/month,start\_time,end\_time,userid,actid,title,priority*  
Notes:

* + 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 the activity can be computed from start and end times.
  + The entry of the activity is only made for its starting hour in the calendar.

Your program should be able to read this file and load the calendar in format described in (2). The program should save the calendar in the exact same format when asked to do so.

Randomly generated sample files (right-click and "save as"): [sample1.txt](http://www.sarimbaig.com/cp/projects/sample1.txt), [sample2.txt](http://www.sarimbaig.com/cp/projects/sample2.txt)

1. **How to Print the Calendar**

This is a console application without a graphical interface. However, the calendar must be printed in the [standard format](http://upload.wikimedia.org/wikipedia/commons/thumb/7/70/2014_calendar.svg/744px-2014_calendar.svg.png). The numbers should be printed in exactly the same block format, with the month names and letters on top for the day names, etc. The numbers for days without activities should be white on the black console. Numbers for days with one of more activities will be light-green on the black console.

1. **Features and Operations**

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 to one or more C++ functions.

|  |
| --- |
| **0) Clean the Calendar** |
| Input: command  Output: If the same user has multiple activities in the same slot, remove all but the most important (heighest priority) of these activities. This is possible because data files are generated at random. |
| **1) List all activities of a given user during a time period.** |
| Input: user id, start date (day and month), end date  Output: line by line listing of activities |
| **2) List the top 5 activities of a given user during a time period.** |
| Input: user id, start date, end date  Output: line by line listing of activities |
| **3) List the five busiest users in a given time period (users with most activities).** |
| Input: start date, end date  Output: Line by line list of user ids, with their corresponding number of activities inside the given time period. |
| **4) Print activity stats for a given month.** |
| Input: month (name or number)  Output: total number of activities in the month, average number of activities per day, the busiest user of the month, number of activities of the busiest user, the busiest day of the month, number of activities in the busiest day |
| **5) Print Calendar Stats (for the whole year).** |
| Input: command  Output: total number of activities in the year, average number of activities per month, the busiest user in each month, number of activities of the busiest users of each month, the busiest month of the year, number of activities in the busiest month |
| **6) Remove a user from the calendar (remove all activities of this user).** |
| Input: user id  Output: data removed |
| **7) Save the calendar.** |
| Input: command  Output: calendar saved |

**Plagiarism Policy**

We will check the codes for plagiarism with each other, and codes from last year's class. Where detected, plagiarism is punishable by awarding zero in all assignments [25% absolute] at least.

**Good Luck! **

Today's Best Online Deals.

**Ads By I - Cinema**