Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Fall 2022

Homework 3 – Time management with binary search trees

Due: 14/11/2022, Monday, 11:55

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You HAVE TO write down the code on your own.
You CAN NOT HELP any friend while coding.
Plagiarism will not be tolerated!!

Introduction

In this homework, you will build a simple time management program to help users split their day between activities. You will construct this program using a binary search tree. Users will tell your program which activity they will do and at what time. Your application will be able to do the following for the user:

1. Show all the activities of the user in chronological order.
2. Tell the user what they will be doing at certain times, and the duration of each activity.
3. Tell the user all the times they will be doing certain activities in chronological order.

The rest of the document will describe the data structure you will use to implement the homework, and the exact operations you will implement.

Similar to the last homework, you will be provided with a "`main.cpp`" file containing a main program, and a header file "`activity_bst.h`" containing a struct definition and declarations of the functions you must implement. Your job is to write a new `.cpp` file containing the implementations of the functions in the "`activity_bst.h`" header file. You are allowed to write additional functions in the `.cpp` file that you will write. You only need to submit the `.cpp` file that you wrote (not the `main.cpp` or the `activity_bst.h` files.) Check the Submission rules for the naming guidelines.

Data structure

You will store the activities that the user will do during the day and their times in a binary search tree. The tree will be sorted based on the times of the activities. In the remainder of the document, we will refer to this binary search tree as the "activity tree". The following is an example of an activity tree:

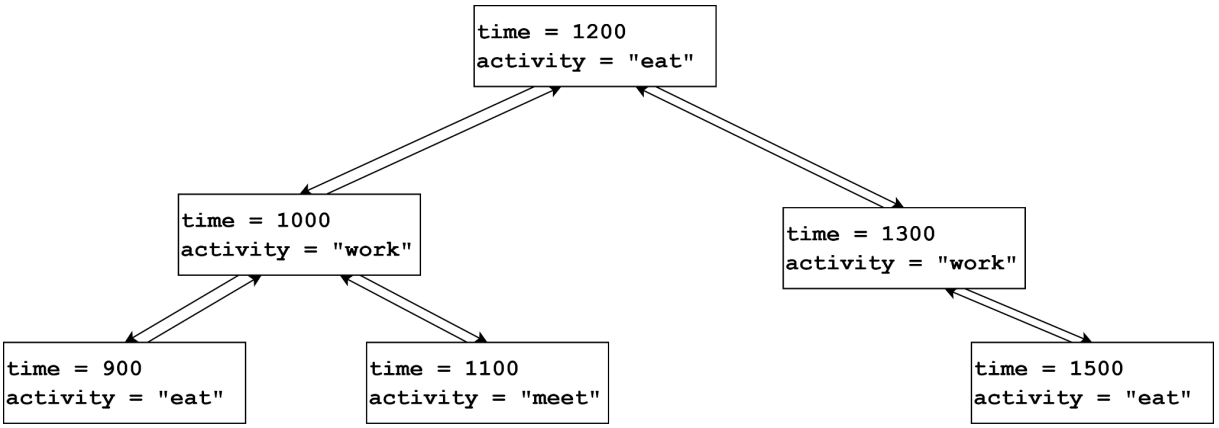

Figure 1: an activity tree representing a daily schedule. Each node contains an integer `time` representing the time of the activity in 24-hour notation (without the colon), and a string `activity` containing the activity name. The tree is a binary search tree sorted according to `time` integers.

The tree nodes are defined as follows:

```
struct tree_node{
   int time; // activity time in 24-hour notation
   // 0 -> 00:00, 1000 -> 10:00, 1312 -> 13:12
   string activity; // activity name
   tree_node* right;
   tree_node* left;
   tree_node* parent;
};
```

Each tree node contains an integer `time` with the time of the activity, a string `activity` with its name, pointers at the node's right and left children, as well as a pointer at the parent of the node. If a node does not have a parent, its parent pointer should be set to nullptr.

Since this tree is a binary search tree, it must be sorted based on some value. **The nodes in this tree will be sorted based on the time integer.**

Rules

Before getting into the functions you will implement, let's lay down some rules.

1. You are allowed to implement additional functions in the `.cpp` file you will write if they help you with your solution.
2. You are NOT allowed to use vectors, arrays, or any other containers to store values anywhere in the code.
3. You are allowed to use libraries within the C++ standard library. These are libraries like `<string>`, `<sstream>`, `<time>`, etc.
4. You are NOT allowed to use external libraries that are not in the C++ standard library. For example, the `strutils.h` library from CS 201.
5. In this homework, we will provide you with a `main.cpp` with a main function and an `activity_bst.h` header file. You are NOT allowed to change these files when submitting your code. While solving the homework, you can do with these files what

you want. However, know that when grading, we will use our own version of these files, and if your code does not work with the files attached with the homework document, it will not work with the files we use for grading.

6. Do not use turkish letters in the code or comments. It causes some issues with the grading software.

Time representation

In the homework, we represent time in 24-hour notation without a colon. So 10:00 am would be 1000, and 2:30 pm would be 1430. The range of legal times is 0-23:59 (0-2359). While writing your functions, you will be asked to print the time with a colon (i.e., print 2359 as "23:59"). For this purpose, we provide the `number_to_time()` function below, which takes time as an integer and returns a string with that number in the form "HH:MM". (The function `fill_with_zeros` is being used by `number_to_time`.) You may use this function in your code.

```
string fill_with_zeros(int num_zeros, const string& input){
   int zeros_to_add = num_zeros - input.length();
   if (zeros_to_add < 0) zeros_to_add = 0;
   return string(zeros_to_add, '0') + input;
}

string number_to_time(int number){
   return fill_with_zeros(2, to_string(number/100))+":"+ fill_with_zeros(2,
to_string(number%100));
}
```

Time ranges

In most of the functions you need to implement, you will have to check whether a certain time point falls in the duration of an activity. Here we define the duration of an activity as the time it starts, up to and excluding the time the next activity starts. So, for the example in Figure 1, the activity "meet" has the duration 11:00-11:59.

As for the time before any activity starts, that time is called "free". So in Figure 1, the activity "free" has the duration 00:00-08:59.

So, if we ask this tree what activity is happening at 07:00, the answer would be the "free" activity, and if we ask it what activity is taking place at 12:00, the answer would be "eat".

Of course, if there is an activity at time 00:00 then there will not be a "free" activity.

As for the last activity, it will always end at 23:59. So for the tree in Figure 1, the activity "eat" which starts at 15:00 has the duration 15:00-23:59.

Functions to implement

Your job will be to implement a few functions that will allow the main program we provide to be executed successfully. The functions you will implement will be described in this section.

Note that for some of the functions, you will be taking a string input (for the input activity). You do not have to do any error checking on this string – our main function will handle that. You do, however, need to do error checking on the time integer. Details about error checking are given in each of the sections below.

add_activity(tree_node*& root, int time, const string& activity)

This function will take a tree_node pointer at the root of a tree, an integer representing a time, and a name for the activity, and will add a tree node to the tree representing this time. Remember that this is a binary search tree sorted based on the time integer.

Error checking

You do not need to make any error checks for the `activity` string. However, you must check that the `time` value passed is legal. A time value is legal if it satisfies two conditions:

1. The minutes part is in the range 0-59.
2. The hours part is in the range 0-23.

Assumptions

You may assume that there will never be two calls to "add_activity" with the same time value. In other words, once an activity is added at time X, no other activities will be added with the same time value.

Output

If activity "act" at time "HH:MM" is added successfully, then the function will print the following string:

```
Added activity 'act' at HH:MM
```

If the activity wasn't added successfully (due to illegal time value) you must print the following string:

```
ERROR! Could not add activity 'act' due to illegal time value
```

print_activity(tree_node* root, int time)

This function will take a pointer at the root of an activity tree and a time value, and will print the activity that will be taking place at that specific time. If no activity is going to take place at that time, it will print the activity as 'free'.

Error checking

You must check that the time value is legal. Follow the same time legality rules described in the `add_activity` function's section.

Output

If the time value is legal, then you will print the following string:

```
Time: HH:MM, Activity: act
```

Where `HH:MM` is the time passed by the user, and `act` is the name of the activity taking place at that time. If no activity is taking place at the time HH:MM, print the activity as `free`.

Please keep in mind the Time ranges rules described in the Section "Time ranges" when determining if a time point falls inside some activity.

For example, given the tree in Figure 1, if we make the call print_activity(root, 1201), you will print:

```
Time: 12:01, Activity: eat
```

And if we make the call print_activity(root, 859), you will print:

```
Time: 08:59, Activity: free
```

If the time value is not legal, print the following string:

```
ERROR! Could not print activity at specific time due to illegal
time
```

print_activity_and_duration(tree_node* root, int time)

This function will take some time value and, if that time value is legal, it will print the activity taking place at the time as well as that activity's start time, and the time the next activity starts.

Error checking

You must check that the time value is legal. Follow the same time legality rules described in the `add_activity` function's section.

Output

If the time value is legal, you will print the following string

```
Time period: [HH:MM - H'H':M'M']: act
```

Where `HH:MM` is the time the activity starts, `H'H':M'M'` is the time the next activity starts (00:00 if there are no other activities starting after this one), and `act` is the name of the

activity. If the time passed to the function is before the start of the first activity of the day, the activity name is `free`.

For example, given the tree in Figure 1, if we make the call print_activity(root, 1201), you will print:

```
Time period: [12:00 - 13:00]: eat
```

`12:00`. is the start time of the activity “eat” and `13:00` is the start time of the next activity after “eat”.

And if we make the call print_activity(root, 859), you will print:

```
Time period: [00:00 - 09:00]: free
```

`00:00` is the start time of the activity “free” (i.e. before any activities start) and `09:00` is the start time of the next activity after “free” (time of the first activity).

And if we make the call print_activity(root, 2300), you will print:

```
Time period: [15:00 - 00:00]: eat
```

`15:00` is the start time of the activity “eat” (i.e. last activity) and `00:00` is the start time of the next activity after “eat” (since this activity is the last activity, the next activity is the beginning of the next day, i.e. `00:00`).

Time range rules are described in the section “Time ranges”.

print_single_activity(tree_node* root, const string& activity)

This function will print the durations of all occurrences of the activity “activity” in chronological order. So, if an event occurs multiple times, all of its durations will be printed from the first one to occur during the day, to the last one.

Error checking

You do not need to do any error checking on the string activity.

Output

For each node with activity== ‘activity’, you will print the time duration of that activity. You must print these durations *in chronological order.* Meaning that you must print the earlier occurrences before the later ones. Each occurrence print will be on a separate line and will have the same format as the one for the function print_activity_and_duration. If the activity never occurs, then we won’t print anything.

For example, for the tree in Figure 1, if we make the call print_single_activity(root, “work”) we would print the output:

```
Time period: [10:00 - 11:00]: work
Time period: [13:00 - 15:00]: work
```

And if we make the call print_single_activity(root, “meet”):

```
Time period: [11:00 - 12:00]: meet
```

If we make the call print_single_activity(root, “free”), we would print:

```
Time period: [00:00 - 09:00]: free
```

And if we make the call print_single_activity(root, “sleep”) we would not print anything.

print_all_activities(tree_node* root)

This function will print all the activities taking place during the day with their start times.

Error checking

There is no error checking you need to do for the input.

Output

You will print the start times of all the activities in the tree, each on a separate line of the form:

```
[HH:MM] - act
```

where `HH:MM` is the start time of the activity, and `act` is its name.

The activities printed do not include the “free” activity; only print the start times of activities that the user adds. If there are no activities added by the user, don’t print anything.

For the tree in Figure 1, if we make the call print_all_activities(root), you will print:

```
[09:00] - eat
[10:00] - work
[11:00] - meet
[12:00] - eat
[13:00] - work
[15:00] - eat
```

delete_tree(tree_node* root)

This function will delete the tree pointed at by root.

Program flow

As mentioned above, you will be provided with a main.cpp file containing a main program that uses the activity tree, as well as a header file containing the struct definition of tree_node and the declarations of the functions you must implement. The following is the main program:

```
int main() {
   tree_node *root = nullptr;
   string user_input;
   cout << "Welcome to your time manager. Enter a command: ";
   while (getline(cin, user_input)) {
       stringstream ss(user_input);
       int action;
       int time;
       string activity;
       ss >> action;
       if (ss.fail()) {
           cout << "ERROR! Action entered is not a number\n";
       } else {
           switch (action) {
               case 0: // Add an activity
                   cout << " **** Add activity ****" << endl;
                   ss >> time >> activity;
                   if (ss.fail()) {
                       cout << "ERROR! Time or activity is invalid\n";
                   } else {
                       add_activity(root, time, activity);
                   }
                   break;
               case 1:
                   cout << " **** Check activity ****" << endl;
                   ss >> time;
                   if (ss.fail()) {
                       cout << "ERROR! Time is invalid\n";
                   } else {
                       print_activity(root, time);
                   }
                   break;
               case 2:
                   cout << " **** Check activity and duration ****" << endl;
                   ss >> time;
                   if (ss.fail()) {
                       cout << "ERROR! Time is invalid\n";
                   } else {
                       print_activity_and_duration(root, time);
                   }
                   break;
```

```
                case 3:
                    cout << " **** Check all occurrences of an activity ****"
                         << endl;
                    ss >> activity;
                    if (ss.fail()) {
                        cout << "ERROR! Activity is invalid\n";
                    } else {
                        print_single_activity(root, activity);
                    }
                    break;
                case 4:
                    cout << " **** Print all occurrences of an activity ****"
                         << endl;
                    print_all_activities(root);
                    break;
                default:
                    cout << "Action not recognized" << endl;
            }
        }
        cout << "--------------------" << endl << endl;
        cout << "Enter a command: ";
    }
    delete_tree(root);
    return 0;
}
```

Sample Runs

The following are sample executions. Your output should match this output exactly. Bolded and italicized text is user input.

Sample run 1

```
Welcome to your time manager. Enter a command: 1 900
 **** Check activity ****
Time: 09:00, Activity: free
-------------------

Enter a command: 2 1000
 **** Check activity and duration ****
Time period: [00:00 - 00:00]: free

-------------------

Enter a command: free
 ERROR! Action entered is not a number
-------------------

Enter a command: 3 free
 **** Check all occurrences of an activity ****
[00:00 - 00:00] - free
-------------------

Enter a command: 4
 **** Print all occurrences of an activity ****
-------------------

Enter a command: ctrl+c
```

Sample run 2

```
Welcome to your time manager. Enter a command: 0 -100 sleep
 **** Add activity ****
ERROR! Could not add activity 'sleep' due to illegal time value
-------------------

Enter a command: 0 2400 sleep
 **** Add activity ****
ERROR! Could not add activity 'sleep' due to illegal time value
-------------------

Enter a command: 0 1060 sleep
 **** Add activity ****
ERROR! Could not add activity 'sleep' due to illegal time value
-------------------

Enter a command: 0 0 sleep
 **** Add activity ****
Added activity 'sleep' at 00:00
-------------------

Enter a command: 1 1000
 **** Check activity ****
Time: 10:00, Activity: sleep
-------------------

Enter a command: 2 12
 **** Check activity and duration ****
Time period: [00:00 - 00:00]: sleep
-------------------

Enter a command: 3 free
 **** Check all occurrences of an activity ****
-------------------

Enter a command: 3 sleep
 **** Check all occurrences of an activity ****
Time period: [00:00 - 00:00]: sleep
-------------------

Enter a command: 4
 **** Print all occurrences of an activity ****
```

```
[00:00] - sleep
-------------------

Enter a command: ctrl+c
```

Sample run 3

```
Welcome to your time manager. Enter a command: 0 1200 eat
 **** Add activity ****
Added activity 'eat' at 12:00
-------------------

Enter a command: 0 1000 work
 **** Add activity ****
Added activity 'work' at 10:00
-------------------

Enter a command: 0 1100 meet
 **** Add activity ****
Added activity 'meet' at 11:00
-------------------

Enter a command: 4
 **** Print all occurrences of an activity ****
[10:00] - work
[11:00] - meet
[12:00] - eat
-------------------

Enter a command: 0 900 eat
 **** Add activity ****
Added activity 'eat' at 09:00
-------------------

Enter a command: 0 1500 eat
 **** Add activity ****
Added activity 'eat' at 15:00
-------------------

Enter a command: 3 eat
 **** Check all occurrences of an activity ****
Time period: [09:00 - 10:00]: eat
Time period: [12:00 - 15:00]: eat
Time period: [15:00 - 00:00]: eat
-------------------

Enter a command: 2 1230
 **** Check activity and duration ****
```

```
Time period: [12:00 - 15:00]: eat
-------------------

Enter a command: 4
 **** Print all occurrences of an activity ****
[09:00] - eat
[10:00] - work
[11:00] - meet
[12:00] - eat
[15:00] - eat
-------------------

Enter a command: 0 1300 work
 **** Add activity ****
Added activity 'work' at 13:00
-------------------

Enter a command: 1 700
 **** Check activity ****
Time: 07:00, Activity: free
-------------------

Enter a command: 2 300
 **** Check activity and duration ****
Time period: [00:00 - 09:00]: free

-------------------

Enter a command: ctrl+c
```

Submission rules

In order to get full credit, your programs must be efficient and well presented, the presence of any redundant computation or bad indentation, missing comments, or irrelevant comments are going to decrease your grades. You also have to use understandable identifier names and informative prompts. Modularity is also important; you have to use functions wherever needed and appropriate.

When we grade your homework we pay attention to these issues. Moreover, in order to observe the real performance of your codes, we will run your programs in Release mode and we will test your programs with very large test cases.

What and where to submit (PLEASE READ, IMPORTANT)

You must write your solution in C++. It'd be a good idea to write your name and last name in the program (as a comment line of course). Submission guidelines are below. Some parts of the grading process are automatic. Students are expected to strictly follow these guidelines in order to have a smooth grading process. If you do not follow these guidelines, depending on the severity of the problem created during the grading process, 5 or more penalty points are to be deducted from the grade.

Name your cpp file that contains your `main` function as follows:

```
SUCourseUserName_YourLastname_YourName_HWnumber.cpp
```

Your SUCourse user name is your SUNet username that is used for your sabanciuniv e-mails. Do NOT use any spaces, non-ASCII and Turkish characters in the file name. For example, if your SUCourse username is cago, your first name is Taha Çağlayan, and your last name is Özbugsizkodyazaroğlu, then the file name must be:

```
cago_Ozbugsızkodyazaroglu_TahaCaglayan_1.cpp
```

If your solution contains other code files, you don’t have to change the other files’ names. Only the file with the main function needs to have the naming convention above.

You shouldn’t add any other files besides your code to the submission folder. In other words, don’t add the example inputs and outputs to the submission.

Place all of your code files inside a folder named with the same naming convention shown above (without the .cpp extension, of course). So, the same student above would place all of his code inside a folder named:

```
cago_Ozbugsızkodyazaroglu_TahaCaglayan_1
```

Compress this folder using a compression program such as WinZip or WinRAR. Please use "zip" compression. "rar", “7z” or any other compression mechanisms are NOT allowed. Our homework processing system only works with zip files. Therefore, make sure that the resulting compressed file has a zip extension. Check that your compressed file opens up

correctly and it contains all your code files. You will receive no credits if your compressed folder does not expand or it does not contain the correct files. The name of the zip file follows the same convention. The zip file for the homework submission by the student mentioned above would be:

```
cago_Ozbugsızkodyazaroglu_TahaCaglayan_1
```

Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.). Successful submission is one of the requirements of the homework. If for some reason you cannot successfully submit your homework and we cannot grade it, your grade will be 0.

Good Luck!

Amro