CS201 – Spring 2015-2016 - Sabancı University

Homework #6: Bus Database Search

Due May 4, Wednesday, 19:00 (Sharp Deadline)

Brief Description

First let us start with a brief definition of the Database concept. A database is a collection of data records stored in a computer in a systematic way. In this way, a computer program can be used to answer queries about the data stored in the database by searching the records in a systematic fashion.

In this homework, you will write a program that will search bus route and time databases to find departure times for given inputs. The database records that you will deal with are stored in two text files (see "Structure of Database (Input) Files" section for details). Your program will basically search and return bus times and destinations for given departure point after a certain time. Resulting list will be stored in a separate output file (see "Name and Structure of Output File" section for details).

After successfully creating the output file, your program will ask parameters for another search until "EXIT" is entered as a termination command for departure location.

After a termination command, your program will close. You should check the "Sample Runs" below.

VERY IMPORTANT!

Your programs will be compiled, executed and evaluated automatically; therefore, you should definitely follow the rules for prompts, inputs and outputs. See **Sample Runs** section for some examples.

- **Order of inputs and outputs** must be in the abovementioned format.
- **Prompts before inputs and outputs** must be **exactly the same** with examples.

Following these rules is crucial for grading, otherwise our software will not be able to process your outputs and you will lose some grades in the best scenario.

IMPORTANT!

If your code does not compile, you will get **zero**. Please be careful about this and double check your code before submission.

Input Check

There are three input checks in this homework:

- First you will ask for name of the file that contains the routes database. If no file will be found, you should print out an error message and then ask for another file name for the routes database.
- The above rule applies for the times database as well.
- The last input check is on the format of the departure time. Time should be in format hh:mm, *hh* for 2-digit hour and *mm* for 2-digit minutes. Example: 03:05 not 3:5 or not 3:05. If this input is wrong, then your program should display an error message and ask for another departure time. (See Sample Runs for examples).
  Please NOTE that this time specific input check is only necessary for user entered inputs and not for the times database file. In the times database file, you may assume that all departure times are in the correct time format.

Structure of Database Files

As mentioned before, the database is stored in two different text files. One is only for route records, the other one is for route-time pair records. These two files will be input files to your program. (i.e. your program will read some data from both files)

Name of both input files will be entered by the user of your program. At the beginning of your program, user will be asked to enter an input file name for the route records. If the user enters a name for a file that does not exist, your program should display an appropriate message and ask for the name of a new input file name until the file is successfully opened. After the first input file is opened successfully, your program will ask for another input file name for time records file. The same file opening control will be performed for this file too. If user enters a name for a file that does not exist, your program should display an appropriate message and ask for the name of a new input file name until the file is successfully opened.

Route Database File

The routes input file contains several information fields for each route. For a single route, there are three columns of information on the same line. Each line corresponds to the record of a route. The record structure for each route is described as follows:

```
routeID routeStart routeEnd
```

Record line starts with a ***routeID***. *routeID*'s are integer values and they are unique in each line. You will need to store this information to match the routes to times in the next file. It is followed by two strings: ***routeStart*** and ***routeEnd***.

The abovementioned record structure, which contains 3 different data items in one line, is just for one route. The input file contains information about several routes. Therefore, there are several such records in the input file. You may assume there is a record in each line of input file, no line is empty. Also please note that the database file is not sorted (ordered).

Times Database File

The times database file contains time-route pairs written by that time. For a single time record, there are 3 pieces of information on the same line. The record structure for each route-time pair is described as follows:

routeID	***departureTime***

routeID is a single integer value.
departureTime is a string value. It is in the format 00:00. (ex: 03:02, 16:45 etc.)

We assume that **no** time-route pair appears **more than once** in the Times database file. For example, if there is a pair such as ***Route11 Time1***, then there will **not** be another pair such as ***Route11 Time1***.

There is no specific order of records in both input files. That means you cannot assume that the records of the input files are ordered according to **r*outeID***, ***departureTime*** or any other data attribute.

The structure of the input files and the associated rules and assumptions explained above are fixed such that you cannot change them or make any other assumptions. Y**ou can assume that the input files will not be empty.** You may examine the sample input files provided in the zip package for this homework (hw6.zip). Please note that we have multiple input files for multiple scenarios, therefore your code should also work with all these possible outcomes.

Important! There might be any number of white spaces (i.e. blanks) and/or tabs between and after each data item in both files. You should handle it.

Database Search/Filter

Database search will basically be searching the routes database file for all routes after a given departure time. First you need to specify a Departure Location for your search. After entering departure location, (as long as it is not EXIT) you will be asked to enter a Departure Time to search for. You should only output routes that are equal to or after this time. For example, if there are three bus routes from canakkale at times: 10:00, 12:00, 14:00 and the departure time is given as 12:00; then you should only print out the routes at 12:00 and 14:00 to the output file.

Your results will contain **all** routes possible from the given departure location after (or equal to) the given departure time.

To do so, you must first search routes.txt and check each *routeStart* value to see if it is equal to the departure location given as input. If a match is found, you should store *routeID* and *routeEnd*. Then you should check times database file to check if there is a corresponding bus route with that stored *routeID*. If a record is found, then you should check the time on that record with the user input departure time. If the route time is later than or equal to the departure time, then you should put a line to your output file as specified in Section: ***Name and Structure of Output File*** below.

When the search is finished, display a message (to the screen) that informs the user about the end of the process and the name of the output file. The message format is as follows:
"The search results are in the file: *output_file_name*"

After successfully creating the output file, your program will ask parameters for another search until "EXIT" word is entered as a termination command for departure location.

After a termination command, your program will close. You should check the "Sample Runs" below.

Name and Structure of the Output File

There will be a single output file. The name of this file will be in the following format: *departureLocation.txt*. For example, if the departure location is Ankara, then your output file should be called as "Ankara.txt".

As mentioned above, all the search results are stored in a single output file. In this file, you should output one line for each route found. The structure of this line is below:

```
departureLocation routeEnd busTime
```

departureLocation is the user's input, ***routeEnd*** is the extracted from the routes database file, ***busTime*** is extracted from the times database file. There is one blank (space) between each of these entries.

There will be any number of such lines, one for each route found, in the output file.

There is one more important rule about the output file content. In the output file, the words that make up these names must be separated by only one white space. (Please see the sample output files provided below and in the zip package of this homework for some examples.)

Some Hints

Maybe you have already realized that the simplest way of processing the input file is reading the data line by line and parsing each line. In order to perform such a processing, the ideal mechanism is to use a combination of `getline` function and input string streams (`istringstream`). We have given/will give examples of such a processing in the lecture and recitations.

Just a reminder: when you are done with an output file, do not forget to **close** it.

Note that string comparisons are **case sensitive**. That means *turkey* and *Turkey* are not equal to each other.

Please see sample runs for examples.

No abrupt program termination please!

You may want to stop the execution of the program at a specific place in the program. Although there are ways of doing this in C++, it is not a good programming practice to abruptly stop the execution in the middle of the program. Therefore, your program flow should continue until the end of the main function and finish there.

Sample Run

Below, we provide a sample run of the program that you will develop. The italic and bold phrases are inputs taken from the user. You should follow the input order in these examples and the prompts your program will display must be **exactly the same** as in the following examples.

Command Line Window

```
Please enter a filename for route database: r
cannot open routes database file
Please enter a filename for route database: routes
cannot open routes database file
Please enter a filename for route database: routes.txt
Please enter a filename for time database: time.txt
cannot open times database file
Please enter a filename for time database: times.txt
Please enter departure location: Istanbul
Please enter start time of travel: 03:45
The search results are in the file: Istanbul.txt
Please enter departure location: Adana
Please enter start time of travel: 1:15
Time is not in the correct format
Please enter start time of travel: 01:15
The search results are in the file: Adana.txt
Please enter departure location: Izmir
Please enter start time of travel: 14:23
The search results are in the file: Izmir.txt
Please enter departure location: Trabzon
Please enter start time of travel: 23:00
The search results are in the file: Trabzon.txt
Please enter departure location: Eskisehir
Please enter start time of travel: 10:104
Time is not in the correct format
Please enter start time of travel: 1010
Time is not in the correct format
Please enter start time of travel: 10:10
The search results are in the file: Eskisehir.txt
Please enter departure location: Ankara
Please enter start time of travel: 13:22
The search results are in the file: Ankara.txt
Please enter departure location: gaziantep
Please enter start time of travel: 07:30
The search results are in the file: gaziantep.txt
Please enter departure location: Antalya
Please enter start time of travel: 12:00
The search results are in the file: Antalya.txt
Please enter departure location: EXIT
Press any key to continue . . .
```

After this Sample Run,

- Antalya.txt is empty, because Antalya is not a departure location in these input files.
- gaziantep.txt is empty, because in input files it is written with an upper case ‘G’ and string comparisons are case sensitive.
- Trabzon.txt is empty, because there no buses from Trabzon after given departure time.
- Other output files are not empty, you may find them in your homework zip package, please check them.

General Rules and Guidelines about Homeworks

The following rules and guidelines will be applicable to all homeworks, unless otherwise noted.

How to get help?

You may ask questions to TAs (Teaching Assistants) of CS201. Office hours of TAs are at the class website. Recitations will partially be dedicated to clarify the issues related to homework, so it is to your benefit to attend recitations.

What and Where to Submit

Please see the detailed instructions below/in the next page. The submission steps will get natural/easy for later homeworks.

Grading and Objections

Careful about the semi-automatic grading: Your programs will be graded using a semi-automated system. Therefore, you should follow the guidelines about input and output order; moreover, you should also use same prompts as given in the Sample Runs. Otherwise semi-automated grading process will fail for your homework, and you may get a zero, or in the best scenario you will lose points.

Grading:

- Late penalty is 10% off of the full grade and only one late day is allowed.
- **Having a correct program is necessary, but not sufficient to get the full grade. Comments, indentation, meaningful and understandable identifier names, informative introduction and prompts, and especially proper use of required functions, unnecessarily long program (which is bad) and unnecessary code duplications (which is also bad) will also affect your grade.**
- Please submit your own work only (even if it is not working). It is really easy to find out "similar" programs!
- For detailed rules and course policy on plagiarism, please check out http://myweb.sabanciuniv.edu/gulsend/su_current_courses/cs-201-spring-2008/plagiarism/ and keep in mind that

Plagiarism will not be tolerated!

Grade announcements: Grades will be posted in SUCourse, and you will get an Announcement at the same time. You will find the grading policy and test cases in that announcement.
Grade objections: It is your right to object to your grade if you think there is a problem, but before making an objection please try the steps below and if you still think there is a problem, contact the TA that graded your homework from the email address provided in the comment section of your announced homework grade or attend the specified objection hour in your grade announcement.

- Check the comment section in the homework tab to see the problem with your homework.
- Download the .zip file you submitted to SUCourse and try to compile it.
- Check the test cases in the announcement and try them with your code.
- Compare your results with the given results in the announcement.

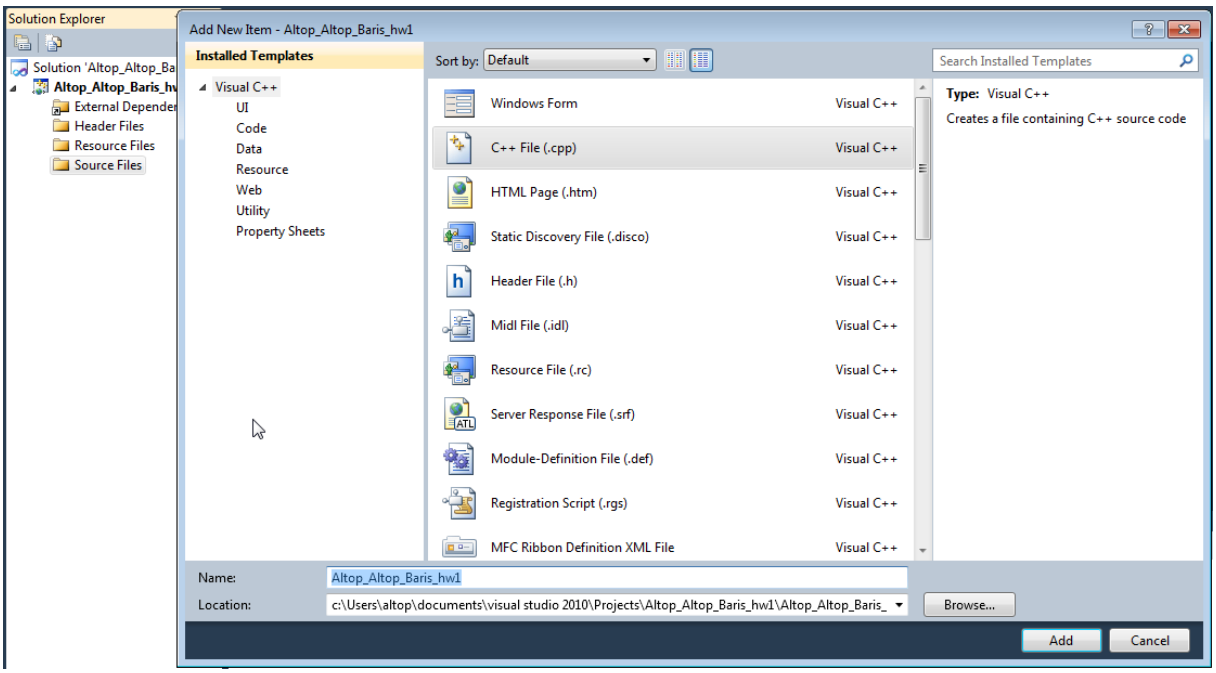
What and where to submit (IMPORTANT)

Submissions guidelines are below. Most 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.

Add your name to the program: It is a good practice to write your name and last name somewhere in the beginning program (as a comment line of course).

Name your submission file:

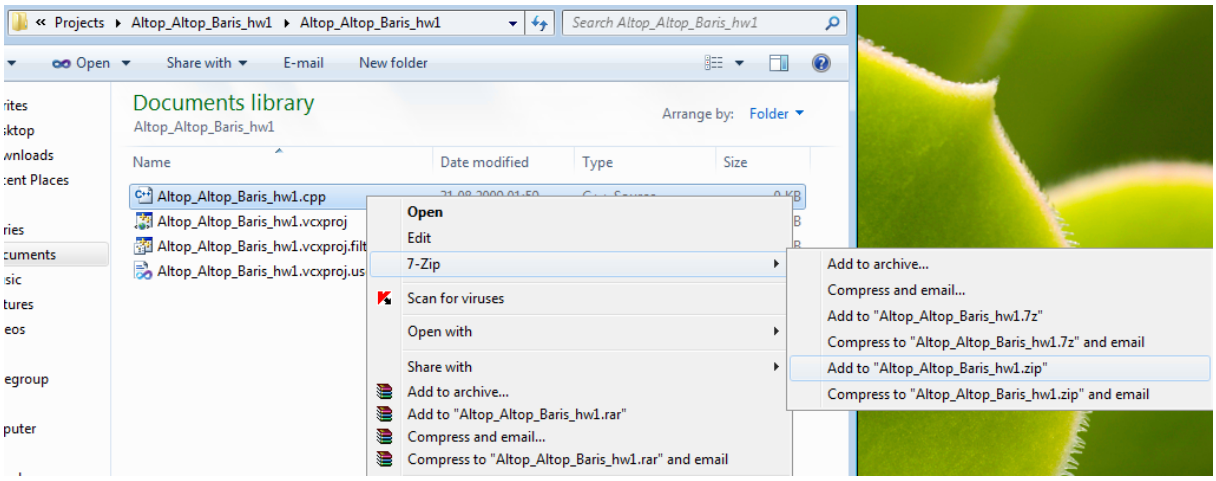
- ❑ Use only English alphabet letters, digits or underscore in the file names. Do not use blank, Turkish characters or any other special symbols or characters.
- ❑ Name your cpp file that contains your program as follows.
  "**SUCourseUserName_YourLastname_YourName_HWnumber.cpp**"



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

Cago_Ozbugsizkodyazaroglu_Caglayan_hw6.cpp

- Do not add any other character or phrase to the file name.
- Make sure that this file is the latest version of your homework program.
- Compress this cpp file using WINZIP or WINRAR programs. Please use "**zip**" compression. "rar" or another compression mechanism is NOT allowed. Our homework processing system works only 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 your **cpp** file. You will receive no credits if your compressed zip file does not expand or it does not contain the correct file.
- The naming convention of the zip file is the same as the cpp file (except the extension of the file of course). The name of the zip file should be as follows.

"**SUCourseUserName_YourLastname_YourName_HWnumber.zip**"

For example zubzipler_Zipleroglu_Zubeyir_hw6.zip is a valid name, but hw6_hoz_HasanOz.zip, HasanOzHoz.zip are NOT valid names.

Submission:

- Submit via SUCourse ONLY! You will receive no credits if you submit by other means (e-mail, paper, etc.).
  1) Click on "Assignments" at CS201 SUCourse (not the CS201 web site).
  2) Click Homework 6 in the assignments list.
  3) Click on "Add Attachments" button.
  4) Click on "Browse" button and select the zip file that you generated.
  5) Now, you have to see your zip file in the "Items to attach" list.
  6) Click on "Continue" button.
  7) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

Resubmission:

- After submission, you will be able to take your homework back and resubmit. In order to resubmit, follow the following steps.
    1) Click on "Assignments" at CS201 SUCourse.
    2) Click Homework 6 in the assignments list.
    3) Click on "Re-submit" button.
    4) Click on "Add/remove Attachments" button
    5) Remove the existing zip file by clicking on "remove" link. This step is very important. If you do not delete the old zip file, we receive both files and the old one may be graded.
    6) Click on "Browse" button and select the new zip file that you want to resubmit.
    7) Now, you have to see your new zip file in the "Items to attach" list.
    8) Click on "Continue" button.
    9) Click on "Submit" button. We cannot see your homework if you do not perform this step even if you upload your file.

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!
Ece Egemen and Gülşen Demiröz