AE1PGA Coursework 4 - Train trips

Introduction

This is the fourth AE1PGA Coursework. It is worth **35% of the module mark**. It requires you to write a program which will calculate the costs of train journeys between cities. The deadline for this exercise is **16:00 on Wednesday 27th of December 2018** .

Read the entire document before beginning the exercise.

If you have any questions about this exercise, please ask in the Q&A forum on Moodle, after a lecture, in a lab, or during the advertised office hours. Do not post your program or parts of your program to Moodle as you are not allowed to share your coursework programs with other students. If any questions requires this exercise to be clarified then this document will be updated and everyone will be notified via Moodle.

Version History

- Version 1.0 - 2018-11-09 - Original version.
- Version 1.1 - 2018-12-01 - Deadline extended to 27th December due to Moodle planned outage.

Submission

You must submit a single C source code file containing all your code for this exercise. This file must be called `train.c` and must not require any other files outside of the standard C headers which are always available. The first line of the file should be a comment which contains your student ID number, username, and full name, of the form:
`// 6512345 zy12345 Joe Blogs`

The file must compile without warnings or errors when I use the command
`gcc -std=c99 -lm -Wall -Wformat -Wwrite-strings train.c -o train`

This command will be run on our Linux server `cslinux`. If it does not compile, for any reason, then you will lose all the marks for testing (common reasons in the past have been submitting a file with the wrong filename, or developing your solution on your personal computer without having tested it on our Linux server). If the file compiles but has warnings then you will lose 50% of testing marks for not correcting the warnings.

The completed source code file should be uploaded to the Coursework 4 Submission link on the AE1PGA Moodle page. You may submit as many times as you wish before the deadline (the last submission before the deadline will be used). After the deadline has passed, if you have already submitted your exercise then you will not be able to submit again. If you have not already submitted then you will be allowed to submit **once**.

Remember that you can only access the Linux server from the designated labs in SEB and

SSB and that these labs are also heavily booked for teaching. **Do not** wait until the last moment to submit as you may find you cannot get into any of the rooms to access your source code files. Not being able to access the machine due to lack of planning is **not** an extenuating circumstance. If you use the VDI or X2Go then the same rules apply. Equipment occasionally breaks down, is full, inaccessible or unreliable. You need to plan ahead to allow time for foreseeable things outside of your control going wrong.

Late submissions: AE1PGA late submission policy is **different** from the standard university policy. Late submissions will lose 5 percentage points **per hour**, rounded up to the next whole hour. This is to better represent the large benefit a small amount of extra time can give at the end of a programming exercise. No late submissions will be accepted more than 24 hours after the exercise deadline. If you have extenuating circumstances you must file them before the deadline.

Plagiarism

You should complete this coursework on your own. Anyone suspected of plagiarism will be investigated and punished in accordance with the university policy on plagiarism (see your student handbook and the University Quality Manual). This may include a mark of zero for this coursework.

You should write the source code required for this assignment yourself. If you use code from other sources (books, web pages, etc), you should use comments to acknowledge this (and marks will be heavily adjusted down accordingly). ***The only exception to this is the prompt function or the dynamic data-structures developed during the lectures; you may use these, with or without modification, without penalty <u>as long as you add a comment saying you have taken them from the lectures and saying how you have modified it (or not modified it)</u>. If you do not acknowledge their source in a comment then it will be regarded as potential plagiarism.***

You must not copy or share source code with other students. You must not work together on your solution. You can informally talk about higher-level ideas but not to a level of detail that would allow you all to create the same source code. Do not share computers, storage devices (eg, usb keys), emails, chat conversations, VM's, user accounts, etc — you are responsbile for making sure there is no way another student can accidentally submit your work as their work.

Remember, it is quite easy for experienced lecturers to spot plagiarism in source code. We also have automated tools that can help us identify shared code, even with modifications designed to hide copying. If you are having problems you should ask questions rather then plagiarize. If you are not able to complete the exercise then you should still submit your incomplete program as that will still get you some of the marks for the parts you have done (but make sure your incomplete solution compiles and partially runs!).

If I have concerns about a submission, I may ask you to come to my office and explain your work in your own words.

Marking

The marking scheme will be as follows:

- *Tests (40%):* Your program should correctly implement the task requirements. A number of tests will be run against your program with different input data designed to test if this is the case for each individual requirement. The tests themselves are secret but general examples of the tests might be:
    - Does the program work with the example I/O in the question?
    - Does the program work with typical valid input?
    - Does the program correctly deal with input around boundary values?
    - Does the program correctly deal with invalid values?
    - Does the program handle errors with resources not being available (eg, malloc failing or a filename being wrong)?
    - Does the program output match the required format?

  As noted in the submission section, **if your program does not compile then you will lose all testing marks**.
- *Appropriate use of language features (30%):* Your program should use the appropriate C language features in your solution. You can use any language features or techniques that you have seen in the course, or you have learned on your own, as long as they are appropriate to your solution. Examples of this might be:
    - If you have many similar values, are you using arrays (or equivalent) instead of many individual variables?
    - Have you broken your program down into separate functions?
    - Are all your function arguments being used?
    - If your functions return values, are they being used?
    - If you have complex data, are you using structures?
    - Are you using loops to avoid repeating many lines of code?
    - Are your if/switch statements making a difference, or is the conditions always true or false making the statement pointless?
- *Use of graph data-structure/algorithms (20%):* Your program is required to use a graph representation of the train data. Is that implementation designed correctly? Is the choice of data-structure and algorithm appropriate, correct, and efficient? This is assessed separately from the tests and general language features sections to focus specifically on your understanding and implementation of the relevant data-structures and algorithms.
- *Source code formatting (10%):* Your program should be correctly formatted and easy to understand by a competent C programmer. This includes, but is not limited to, indentation, bracketing, variable/function naming, and use of comments. See the lecture notes, and the example programs for examples of correctly formatting programs.

Late Submissions: see submission section above.

Task

Your task is to write a program that will check the route and price of train tickets between stations. A representation of rail connections between stations will be stored in a file that your program will need to load. The filename will be specified as a command line argument. Your program should load this file and use it to make a graph representation of the train network. It should then prompt the user to enter the names of the start and end stations, and then use some graph algorithm of your chosing to decide the shortest route between the stations, and display it to the user. It should repeatedly do this until the user wants to quit the program.

If the program is run without the correct number of command line parameters, it should exit with the error message "Invalid command line arguments. Usage: train <disances.txt>". The filename may be a relative or absolute pathname which can be understood by `fopen`. If the file cannot be opened, use `perror` to print the error message "Cannot open file." together with the operating system specific error message and exit with exit code 1.

The format of the distance file is as follows:

- Lines are separated by a single newline character '\n'.
- On each line, there are multiple cells. Each cell is separated by a comma character ','.
- Each cell may or may not have a value. Cells without values have nothing between commas or between the start/end of the line and the comma.
- Each value can contain any number of any printable ASCII characters, excluding the comma ',' and newline '\n' characters.
- Blank lines (lines with no characters) at the end of the file should be ignored.
- The first line of values in the cells are organised like a table. The first row are the names of stations. The first column are the name of stations. The top-left cell should be empty. The rest of the values of the cells represents the distance in kilometers from the station of that row to the station of that column (ie, from the station of the left to the station on the top). If the cell is empty, that represents no direct connection between those two stations. A visualisation of an example table is below:
- Files which do not match this format are invalid. In particular, if a distance cell contains something other than a positive non-zero integer or empty then it is invalid.
- In the case of invalid files, the program should print the error message "Invalid distances file." and exit with exit code 2.

Your program should load the data from this file into a graph data-structure, where the stations are vertexes, the connections are directed edges, and distances are edge weights. You will then use this data-structure to calculate the journeys required below. Your program should not have to access the file again beyond this point. Note that connections do not have to be symmetric, ie, there may be pairs of stations where you can travel from A to B but not B to A. You can assume that the graph data will be a *weakly connected directed graph*.

Once the graph has been created, you should prompt the user with the message "Start station: " and read in a string. The user can quit the program by entering nothing for the start station; the program should then exit with exit code 0. Otherwise, if that string is not the name of a station then the program should print the error message "No such station." and prompt for the start station again. All user input should be matched case-sensitively.

After the start station, you should prompt the user with the message "End station: " and read in a string. If that string is not the name of a station then the program should print the error message "No such station." and prompt for the *start* station again. If the start and end station are the same, the program should print "No journey, same start and end station." and prompt for the *start* station again.

If both station names are valid, the program should calculate the shortest journey from the start to the end station. You should chose an appropriate graph algorithm to do this efficiently to give you the data you need to display the output below. The program should print out the journey in the following format:

```
From S
via
C1
C2
C3
...
To E
Distance XXX km
Cost YYY RMB
```

where *S* and *E* are the start and end station names; *C1, C2, C3, ...* are all the intermediate station names that the journey will go through; *XXX* is the total distance of the journey; and *YYY* is the total cost of the journey (see next section). If there are no intermediate stations, then the output should display "direct" instead of "via" (and obviously not print any intermediate stations). Once this has been printed out then it should prompt for the start station again.
If there is no possible journey between the stations then the program should print "No possible journey." and prompt for the start station again.

The cost of the journey is calculated as *(the total distance multiplied by 1.2) + (the number of intermediate stations multiplied by 25)*. If the result is not a whole number then it should be rounded up to the next nearest integer.

If the program needs to exit because it cannot allocate memory, it should print the error message "Unable to allocate memory." and exit with exit code 3. If the program needs to exit for any other reason not covered in this document, print an appropriate error message and exit with exit code 4.

Example input/output

Given the following distances file (distances1.txt, with the file in the current directory):

```
,Ningbo,Hangzhou,Suzhou,Changzhou,Shanghai,Taizhou,Wenzhou,Jinhua,Fuzhou,Nanjing
Ningbo,,155,,,,380,,,,
Hangzhou,155,,,210,180,,,180,,280
Suzhou,,,,95,90,,,,,
Changzhou,,210,95,,,,,,,130
Shanghai,,180,90,,,,,,,
Taizhou,380,,,,,,610,,,
Wenzhou,,,,,,610,,235,325,
Jinhua,,180,,,,,235,,,
Fuzhou,,,,,,,325,,,
Nanjing,,280,,130,,,,,,
```

Running the program and just pressing return:

```
zlizpd3 $ ./train distances1.txt
Start station:
zlizpd3 $
```

Running the program:

```
zlizpd3 $ ./train distances1.txt
Start station: Ningbo
End station: Suzhou
From Ningbo
via
Hangzhou
Shanghai
To Suzhou
Distance 425 km
Cost 610 RMB
Start station: Ningbo
End station: Ningbo
No journey, same start and end station.
Start station: Glasgow
No such station.
Start station: Nanjing
End station: Glasgow
No such station.
Start station: 341ed admom1 q!!!!
No such station.
Start station: Wenzhou
End station: Fuzhou
From Wenzhou
direct
To Fuzhou
Distance 325 km
Cost 390 RMB
Start station:
zlizpd3 $
```

Running the program with an invalid file:

```
zlizpd3 $ ./train invalid1.txt
Invalid distances file.
zlizpd3 $
```

Running the program with a wrong filename:

```
zlizpd3 $ ./train missing.txt
Cannot open file.
zlizpd3 $
```

Hints

- If you are given a file name, that file might not exist or you might not have permission to read it!
- The filename does not have to be the same as the example and it does not have to be in the same directory as the program or your home directory. Do not hard-code your filename!
- Remember to free any memory which you no longer need. Your program should not have any memory leaks (dynamically allocated areas of memory which are no longer reachable). You will need to consider how the responsibility for allocated data transfers as your program runs.
- On Linux, you can check the exit code of your program by running `echo $?` as the next command after your program has exited.
- The exercise does not need the full 2 weeks given to complete it. You should be able to finish it, including a few problems and debugging, in roughly a single week (about 8 hours non-contact time per week for this module). You have been given more time than that so you can fit it in around your other courseworks and studying. Please plan ahead and do not leave it until the last moment.

END