Tentamen 12 Oktober 2017, vragen

Computation I: hardware/software interface (Technische Universiteit Eindhoven)

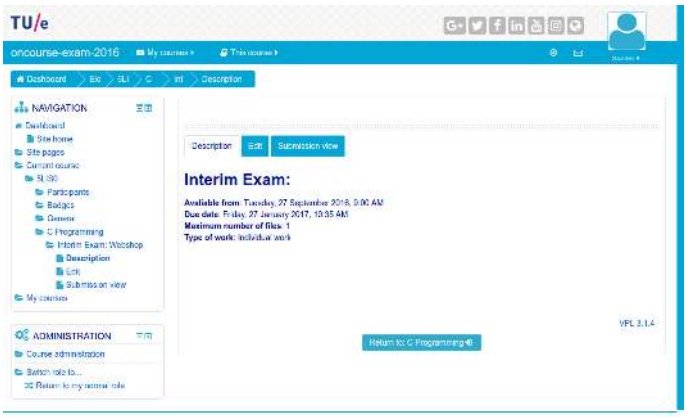
Interim Exam: Trains
12-10-2017, 18.45-20.30

In this interim exam you will develop a program that can be used to find the shortest route (number of stations) between two train stations. The user will be able to add and remove train stations, set destinations that can be reached from a station, and compute the shortest route between stations.

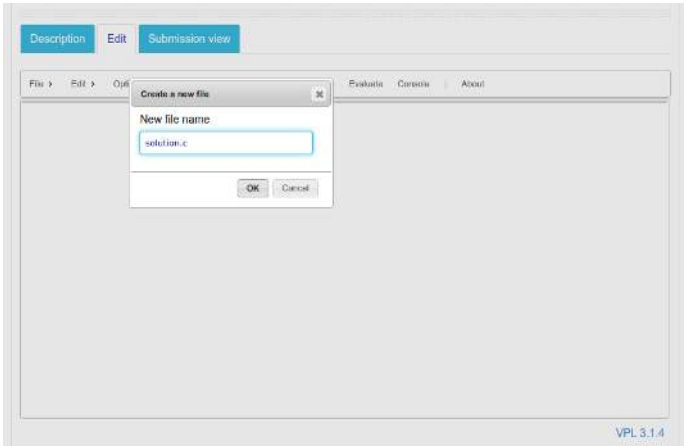
Important

- Your grade is based on the number of cases passed. So try to complete the cases one by one.
- You can press "evaluate" as often as you like during the exam to evaluate your solution. We advise you to do this regularly during the exam.
- The grade is based on your **last** submission. So make sure you submit a working version that completes as many cases as possible.

Task 1. Open the correct C programming exam on exam.oncourse.tue.nl. You will see a screen similar to:

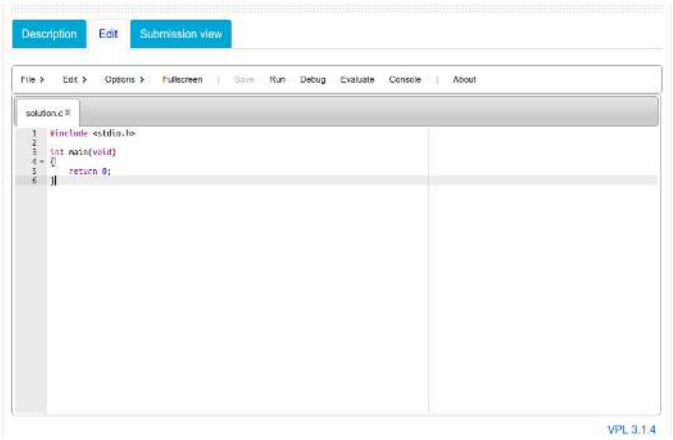


Select the "Edit" tab and the following screen will appear:



Provide a useful name for your C file (e.g., solution.c) and press "Ok".

You will then see the following screen:



You can write your own C program in the text editor that is now shown in your browser. Once you press "Save" you can "Run" and "Evaluate" your program. Using the "Run" command you will see a terminal where you can provide input to your program. You can use this to debug your program. Using the "Evaluate" command all test cases are evaluated and at the end the results are displayed (grade and errors if present).

Task 2. Create a struct of type "`struct station`" that can be used to store the name of the station (type `char *`), a single linked list of destinations (stations that can be reached from this station - type `struct station *`), and a pointer to the next station in the list (type `struct station *`).

Task 3. Write a function "`struct station *findStation(struct station *stations, char *station_name)`" that returns a pointer to the station with the name `station_name` in the list `stations`. If such a station does not exist, the function should return the value `NULL`.

Task 4. Write a function "`void addStation(struct station **stations, char *station_name)`" that adds a new station with the name `station_name` to the front of the list `stations`.

If a station with the name `station_name` already exists in the list `stations`, then the station should not be added again. Instead the following error message should be printed (with the actual name of the station at the position of `station_name`):

```
Station station_name already exists.
```

Task 5. Extend your C program such that it asks the user to select the command that needs to be performed. The commands that need to be supported are listed in the following table:

Character	Command
s	add station
d	add destination
p	print overview of stations and their destinations
r	remove station
f	find shortest path between two stations
q	quit

The output of your program should look as follows:

```
Command (s/d/p/r/f/q): s
Station: Breda
Command (s/d/p/r/f/q): s
Station: Breda
Station Breda already exists.
Command (s/d/p/r/f/q): q
```

Task 6. Write a function "`void addDestination(struct station *station, char *station_name)`" that adds the station with the name `station_name` as a station that can be reached immediately from the station `station`.

Note that a destination only works in one direction. So, when you add the destination Breda to the station Eindhoven, then you can go from Eindhoven to Breda, but not the other way around. Hence, to create a bi-directional connection between these cities you need to call the function `addDestination` at least twice.

A destination should always be added at the front of the list of destinations of a station. Furthermore, a destination should never be added more then once.

Extend the program such that when the user inputs the character 'd' as a command, the user is asked to input two stations a and b and subsequently the destination b is added to station a. The output of your program should look as shown below. Note that the output also contains various error conditions/messages that your program needs to support.

```
Command (s/d/p/r/f/q): s
Station: Breda
Command (s/d/p/r/f/q): s
Station: Eindhoven
Command (s/d/p/r/f/q): d
Station A: Breda
Station B: Eindhoven
Command (s/d/p/r/f/q): d
Station A: Breda
Station B: Eindhoven
Station Eindhoven already a destination for Breda
Command (s/d/p/r/f/q): d
Station A: Tilburg
Station Tilburg does not exist.
Command (s/d/p/r/f/q): q
```

Task 7. Write a function "`void printStations(struct station *stations)`" that outputs the name and destinations that can be reached for each station in the list `stations`.

Extend the program such that when the user inputs the character 'p' as a command, the `printStations` function is executed. The output of your program should now look as follows:

```
Command (s/d/p/r/f/q): s
Station: Breda
Command (s/d/p/r/f/q): s
Station: Eindhoven
Command (s/d/p/r/f/q): s
Station: Den Bosch
Command (s/d/p/r/f/q): d
Station A: Breda
Station B: Eindhoven
Command (s/d/p/r/f/q): d
Station A: Breda
Station B: Den Bosch
Command (s/d/p/r/f/q): d
Station A: Eindhoven
Station B: Breda
Command (s/d/p/r/f/q): p
Station: Den Bosch
Station: Eindhoven
- Breda
Station: Breda
- Den Bosch
- Eindhoven
Command (s/d/p/r/f/q): q
```

Task 8. Write a function "`void removeStation(struct station **stations, char *station_name)`" that removes the station with name `station_name` from the list `stations`. Note that the station should also be removed from the destination of all other stations from which the station could originally be reached.

Extend the program such that when the user inputs the character 'r' as a command, the user is asked to input the name of the station to be removed and subsequently this command is executed using the function `removeStation`. The output of your program should now look as follows:

```
Command (s/d/p/r/f/q): s
Station: Breda
Command (s/d/p/r/f/q): s
Station: Eindhoven
Command (s/d/p/r/f/q): d
Station A: Breda
Station B: Eindhoven
Command (s/d/p/r/f/q): d
Station A: Eindhoven
Station B: Breda
Command (s/d/p/r/f/q): r
Station: Breda
Command (s/d/p/r/f/q): p
Station: Eindhoven
Command (s/d/p/r/f/q): q
```

Task 9. Write a function "`int findShortestPath(struct station *stations, struct station *a, struct station *b)`" that returns the number of stations you need to visit (excluding your departure station) when going from station `a` to station `b` considering all possible stations and their destinations stored in the single linked list `stations`.

The function will make use of recursion and back-tracking. To make this work you need to remember which stations you have visited. A good way to do this is to add an extra variable `int visit` to the datatype `struct station` that you can set/unset when you visit the station during the recursion.

Extend the program such that when the user inputs the character 'f' as a command, the user is asked to input the name of the stations a (start) and b (destination) and subsequently this command is executed using the function `findShortestPath`. The output of your program should now look as follows:

```
Command (s/d/p/r/f/q): s
Station: Breda
Command (s/d/p/r/f/q): s
Station: Eindhoven
Command (s/d/p/r/f/q): d
Station A: Breda
Station B: Eindhoven
Command (s/d/p/r/f/q): f
Station A: Breda
Station B: Eindhoven
Shortest path 1 stations.
Command (s/d/p/r/f/q): f
Station A: Eindhoven
Station B: Breda
No path found.
Command (s/d/p/r/f/q): q
```

Submission: Your final solution must be submitted through OnCourse which will automatically grade this submission. The input and output of the test cases used by OnCourse is not provided on the next page (due to the length of some cases). It will however as always appear when a test case fails on OnCourse.

Input / output test cases

Case 0

Input:

```
s
Breda
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Breda
Command (s/d/p/r/f/q):
```

Case 1

Input:

```
s
Breda
s
Sittard
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Station: Breda
Command (s/d/p/r/f/q):
```

Case 2

Input:

```
s
Breda
s
Eindhoven
s
Sittard
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Station: Eindhoven
Station: Breda
Command (s/d/p/r/f/q):
```

Case 3

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
- Eindhoven
Station: Eindhoven
Station: Breda
Command (s/d/p/r/f/q):
```

Case 4

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
- Eindhoven
Station: Eindhoven
- Breda
Station: Breda
Command (s/d/p/r/f/q):
```

Case 5

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Sittard
Eindhoven
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Command (s/d/p/r/f/q): Station: Sittard
- Eindhoven
Station: Eindhoven
Station: Breda
Command (s/d/p/r/f/q):
```

Case 6

Input:

```
s
Breda
d
Tilburg
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station A: Station Tilburg does
```

Case 7

Input:

```
s
Breda
s
Breda
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Station Breda already
Command (s/d/p/r/f/q):
```

Case 8

Input:

```
s
Breda
s
Eindhoven
s
Sittard
r
Sittard
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Station: Breda
Command (s/d/p/r/f/q):
```

Case 9

Input:

```
s
Breda
s
Eindhoven
s
Sittard
r
Eindhoven
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Station: Breda
Command (s/d/p/r/f/q):
```

Case 10

Input:

```
s
Breda
s
Eindhoven
s
Sittard
r
Breda
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Station: Eindhoven
Command (s/d/p/r/f/q):
```

Case 11

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
r
Eindhoven
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Station: Breda
Command (s/d/p/r/f/q):
```

Case 12

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
r
Breda
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
- Eindhoven
Station: Eindhoven
Command (s/d/p/r/f/q):
```

Case 13

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
r
Sittard
p
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
- Breda
Station: Breda
Command (s/d/p/r/f/q):
```

Case 14

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
f
Sittard
Breda
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Command (s/d/p/r/f/q):
```

Case 15

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
f
Breda
Sittard
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Command (s/d/p/r/f/q):
```

Case 16

Input:

```
s
Breda
s
Eindhoven
s
Sittard
d
Sittard
Eindhoven
d
Eindhoven
Breda
d
Sittard
Breda
f
Sittard
Breda
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Command (s/d/p/r/f/q):
```

Case 17

Input:

```
s
Breda
s
Eindhoven
s
Sittard
s
Den Bosch
d
Sittard
Eindhoven
d
Eindhoven
Breda
d
Eindhoven
Den Bosch
d
Den Bosch
Eindhoven
f
Sittard
Breda
q
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
Command (s/d/p/r/f/q):
```

Case 18

Input:

```
s
B1
s
B2
s
B3
s
B4
s
B5
s
B6
s
B7
s
B8
s
B9
s
B10
s
B11
s
B12
s
B13
s
B14
s
B15
s
B16
s
B17
s
B18
s
B19
s
B20
s
B21
s
B22
s
B23
s
B24
s
B25
s
B26
s
B27
s
B28
s
B29
s
B30
s
B31
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
```

Case 19

Input:

```
s
B1
s
B2
s
B3
s
B4
s
B5
s
B6
s
B7
s
B8
s
B9
s
B10
s
B11
s
B12
s
B13
s
B14
s
B15
s
B16
s
B17
s
B18
s
B19
s
B20
s
B21
s
B22
s
B23
s
B24
s
B25
s
B26
s
B27
s
B28
s
B29
s
B30
s
B31
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
```

Case 20

Input:

```
s
B1
s
B2
s
B3
s
B4
s
B5
s
B6
s
B7
s
B8
s
B9
s
B10
s
B11
s
B12
s
B13
s
B14
s
B15
s
B16
s
B17
s
B18
s
B19
s
B20
s
B21
s
B22
s
B23
s
B24
s
B25
s
B26
s
B27
s
B28
s
B29
s
B30
s
B31
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
```

Case 21

Input:

```
s
Breda
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
Breda
Eindhoven
r
Eindhoven
s
Eindhoven
d
```

Output:

```
Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q): Station: Command (s/d/p/r/f/q):
```