INFO0027: Project 1 (20% – Individual)
Smart Keyboard

Laurent Mathy - François Straet

Submission before April 19

1 Context

You work as a software engineer for the *SNCB*, the famous belgian railway company. They observe longer and longer queues behind each ticket vending machines, yielding frustrating wait times for hurried travelers. They make you responsible for fixing this issue.

Investigating vending machines data, you notice that a significant amount of time is lost, on the one hand due to spelling mistakes while entering the destination station, on the other because the validity check, which happens at each key press, is slow.

Being well trained to data structures and algorithms, you ensure them that you can write a program that will fix the issue.

2 General requirements

In this project, you have to implement a smart keyboard in `C`. A smart keyboard, once initialized with a set of station names, offers the following two features:

1. *completion*: given a string prefix, give all the possible stations that *contain* that prefix,
2. *suggestion*: given a string prefix, it suggests all the possible following letters to continue one of its completions.

Output a single element (i.e., a letter suggestion or a station completion) per line.

3 Implementation

The precise implementation of the program is yours to define, however you must produce well structured code. That is, it should be appropriately split into several files in a meaningful way.

The only restrictions concerns the name, the inputs and the outputs of your program. Your program, named `smartkeyboard`, must comply with the following interface:

```
./smartkeyboard <stations> {complete,suggest} <prefix>
```

Where:

- `<stations>` is the path to a file containing one station name per line,
- `{complete,suggest}` is the command to be performed,
- `<prefix>` is the prefix to run the command on.

The output of your program will contain one element per line, for both *completion* and *suggestion*. Example runs are given in the Appendix.

You are also asked to provide your version of the Makefile skeleton to build your program.

4 Remarks

1. A newline is always marked by the line feed character, `\n` and only by that character (not by a `\r\n`).
2. You can that the input files contain only ASCII alphanumeric characters, plus the hyphen, `'-'` and the space `' '` (no special symbol, no accentuated character).
3. Your system is expected to be case insensitive, i.e. `'A'` is equivalent to `'a'`.
4. If the prefix is itself a whole valid station, e.g. `"Liege-Guillemins"`, the character suggestion is represented by a single dot (`.`) character.
5. For *completion*, you can assume that the provided prefix is at least one character long. For *suggestion*, this assumption does **not** hold.
6. If an error is encountered, it must report it to the standard error and leave the standard input empty.
7. Further questions about the project are welcome on the eCampus forum.

5 Report

You must provide a report of maximum 3 pages. The report should contain a description of your algorithm as well as a performance study, showing the fitness for purpose of your solution to the problem. What to measure, and how, is left to your discretion.

We would also appreciate an estimate of the time you passed on the assignment, and what the main difficulties you encountered were.

6 Evaluation and tests

Your program can be tested on the submission platform. A set of automatic tests will allow you to check if your program satisfies the requirements. Depending on the tests, a **temporary** mark will be attributed to your work. Note that this mark does not represent the final mark. Indeed, other criteria such as the structure of your code, the efficiency, the correctness of other tests and your report will also be considered. In other words, make clean code.

You are however **reminded** that the platform is a **submission** platform, not a test platform.

7 Submission

Projects must be submitted before April 19, 11:59pm.

After this time, a penalty will be applied to late submissions. This penalty is calculated as a deduction of $2^N - 1$ marks (where N is the number of started days after the deadline).

Your submission will include your code (all the required `.c,.h` files and your `Makefile` at the root of the archive), along with your report in pdf format.

Submissions will be made, as a `.tar.gz` or `.zip` archive, on the submission system.

Your code must compile with `gcc`, on the *ms8*** machines, without error or warning (`-Wall -pedantic --std=c99` flags included). Failure to compile will result in an awarded mark of 0. Likewise, warnings and poor spatial or time performance when run over large input will systematically result in lost marks.

Bon travail...

Appendix

In the following, we consider the following file to be present in our working directory:

`stations.txt` with content:

```
Libramont
Liege-Carre
Liege-Guillemins
Liege-Saint-Lambert
Louvain
Louvain-La-Neuve-Universite
```

Also, the exact ordering of the lines in the output does not matter.

Basic example

Call to your program:

```
./smartkeyboard stations.txt suggest Li
```

Expected output:

```
b
e
```

Call to your program:

```
./smartkeyboard stations.txt complete Li
```

Expected output:

```
Libramont
Liege-Carre
Liege-Guillemins
Liege-Saint-Lambert
```

Prefix inside

Call to your program:

```
./smartkeyboard stations.txt suggest Guill
```

Expected output:

```
e
```

Call to your program:

```
./smartkeyboard stations.txt complete Guill
```

Expected output:

```
Liege-Guillemins
```

Call to your program:

```
./smartkeyboard stations.txt suggest m
```

Expected output:

```
o
i
b
```

Call to your program:

```
./smartkeyboard stations.txt complete m
```

Expected output:

```
Libramont
Liege-Guillemins
Liège-Saint-Lambert
```

Empty string as suggestion

Call to your program:

```
./smartkeyboard stations.txt suggest Louvain
```

Expected output:

```
.
-
```

Call to your program:

```
./smartkeyboard stations.txt complete Louvain
```

Expected output:

```
Louvain
Louvain-La-Neuve-Universite
```

Erroneous prefix

Call to your program:

```
./smartkeyboard stations.txt complete Outsiplou
```

or:

```
./smartkeyboard stations.txt suggest Outsiplou
```

Expected output:

```
<Some meaningful error message in stderr and empty stdout>
```