New York University Abu Dhabi
CS-UH 1050 – Spring 2019
Programming Project 1
Out: Tuesday, February 12, 2019
Due: Tuesday, February 26, 2019 11:59 pm

Preamble:

In this project, you will develop a speller application termed `mydictionary`, that can help a user lookup English words in a provided dictionary. You will create your program using the elementary data structures we have discussed in class (mostly lists and arrays) and you will have to use the `Linux/FreeBSD C++` environment.

Overall Description:

Your program should take in its command line, the name of the dictionary to be used as a parameter. Once the `mydictionary` is in operation at its prompt, the user could enquire about specific word(s). Your program should provide three key functionalities, namely, it should:

- tell whether a user-provided word is included in the dictionary used
- list all the words of the dictionary that commence with a user-provided *prefix*
- list all the words of the dictionary that match the input up to one character.

While designing your program you should consider possible alternatives in terms of data structures used to store and search the dictionary. The objective should be to make the *best possible choice* so that you can achieve short response time(s) for the posed queries.

Requirements:

The dictionary file contains one word per line and we assume that the words appear in lexicographic order. Your program will have to determine the precise number of the words in a dictionary at *run-time.*

`mydictionary` should be able to check for (i) complete words (ii) prefixes (iii) words that match the input up to one character.

(i) When searching for full words, your program should report:

- whether a word was found or not.
- the number of word comparisons your program performed.

For instance, if you are looking for the word `arcade` and the word is part of the dictionary in use, the outcome could be:

```
word found
631 word comparisons carried out
```

N.B. `arcade` is a different word from `aRcAde` in case both such entries are included in the dictionary.

(ii) Your prefix search is enabled through the use of what is known as *wildcard* `*` [1]. E.g., search for `explo*`. Here, `mydictionary` should furnish:

- the number of words in the dictionary that match the non-empty prefix,
- a list of these words shown on the `tty`. Actually, you do not need to display all qualifying words but only up to a maximum number as this might be indicated in the command line of `mydictionary`
- the number of comparisons it took to reach the first match found in the dictionary.

[1] this literally means "match any number of characters until the end of the string".

For the aforementioned query example `explo*`, the outcome could be:

```
4 matches found
exploration
exploratory
explore
explorer
204 word comparisons up-to the first match
```

(iii) The third type of input is a word with exactly one character, at position 2 or later, replaced by the wildcard `?`.[2] E.g., search for `explo?e`. Here, `mydictionary` should furnish:

- the number of words in the dictionary that match the input
- a list of these words shown on the `tty`. Actually, you do not need to display all qualifying words but only up to a maximum number as indicated in the command line of `mydictionary`
- the number of comparisons it took to return the entire answer.

For the aforementioned query example `explo?e`, the outcome could be:

```
2 matches found
explode
explore
250 word comparisons executed
```

Your search should clearly avoid searching the entire dictionary for each query. In this context, you should try to come up with a better solution than simply storing all the words in a linked list.

Procedural Matters:

◇ Your program is to be written in `C++` and must run on the `Linux` or `FreeBSD` operating system.
◇ You will have to first submit your project electronically and then, you will have to demonstrate your work.
◇ Khalid Mengal (`khalid.mengal-AT+nyu.edu`) will be responsible for answering questions as well as reviewing and marking the assignment.

How to Invoke your Application:

Your application should be invoked as follows

```
mymachine-promt >>  ./mydictionary -d <dictionaryFile> -l <numOfWordsInOutput>
```

where the flag `-d` indicates that the lexeme that follows is the file name of the dictionary to be used and the `-l` flag indicates the number that follows is the maximum number of results to be printed on the output.

Once the program starts, it presents the user with a prompt. Every time, the user types in a query, your application generates and prints out the result(s) and finally, it should give the prompt again to indicate that it expects the next query. Your program should terminate once the user types at the prompt `exit`.

Group submission:

You should submit your work in groups of two. Both members of a group should be able to answer questions about *any* aspect of the project. To form a group, please send an email to Khalid Mengal **before** submission with the names and NetIDs of the group members. **Both** members of the group should each submit the same set of files by the deadline.

What you Need to Submit:

1. A directory that contains all your work including source, header, `Makefile`, a `README` file, etc.

2. A short write-up about the algorithmic/design choices you made; 2 (at most 3) pages in PDF format are expected.

[2] this literally means "any single character".

3. All the above should be submitted in the form of `tar` or `zip` file bearing your name (for instance `WalterBenjamin-Proj1.tar`).

4. Submit the above tar/zip-ball using *NYUclasses*.

Grading Scheme:

Aspect of Programming Assignment	Percentage of Grade (0–100)
Quality of Code Organization & Modularity	20%
Correct and Efficient Execution of Queries	60 %
Use of Makefile & Separate Compilation	10 %
Well-Commented Code	10 %

Other Noteworthy Points:

1. You have to use *separate compilation* in the development of your program. In other words your code should be distributed into at least two `cpp` files (perhaps more if necessary).

2. We will provide on *NYUclasses* dictionaries files of different sizes.

3. Although it is understood that you may exchange ideas on how to make things work and seek advice from fellow students outside your group, sharing of code is *not allowed*.

4. If you use code that is not your own, you will have to provide *appropriate citation* (i.e., explicitly state where you found the code). Otherwise, plagiarism issues may arise. Regardless, you have to fully understand what and how such pieces of code do. There are tools for detecting network-wide code similarity and we do use them.

5. Don't share you code on public code sharing websites e.g. GitHub.