Assignment

April 4, 2013

Your assignment is to write a translator from *English to Elvish*. You are given the main function in a file `main.cpp`. You must provide an appropriate header file `Translator.h` and an appropriate source file `Translator.cpp` that implements the methods defined in `Translator.h`.

The executable will be generated using the command

```
c++ -o translate.exe Translator.cpp main.cpp
```

The program will be run by executing the command

```
tranlate.exe simpleStory.txt
```

where the argument to the command line (in this case `simpleStory.txt`) is a file of English words which is translated into Elvish and then back to English by the program.

Please note:

1. You *cannot* modify the `main.cpp` file.
2. You are completely free to write `Translate.cpp` and `Translate.h` *any* way you like, provided they compile with the main file, without modifying the main file.
3. To be precise, you can use any advanced parts of `C++` that you may have learned outside the scope of the lectures. But this assignment is also completely solvable using only the concepts that we have covered in class.

By examining `main.cpp` you should be able to discern that there are at least three public methods required to be implemented. These are

1. `Translator::Tranlator(const char filename[])`: This is a constructor that takes a single argument as input, which corresponds to the name of the file containing a dictionary of English words and their Elvish equivalents. The constructor should initialise the object. Exactly what it does depends on how you decide to represent a translator object. You could decide to read all the information in the input file into an array or some other structure, so that it is easy to search for words.
2. `Translator::toElvish(char translatedLine[], const char lineToTranslate[])`: This function takes an input argument `lineToTranslate` which is an array of characters containing a full line of English words; and an output

argument `translatedLine` which is an array of characters in which the English words have been translated into Elvish words.

3. `Translator::toEnglish(char translatedLine[], const char lineToTranslate[])`: This function takes an input argument `lineToTranslate` which is an array of characters containing a full line of Elvish words; and an output argument `translatedLine` which is an array of characters in which the Elvish words have been translated into Englsih words.

Depending on how you decide to implement the `Translator` class, you may need some other methods to support the above methods.

Translation Rules

- All the words in the dictionary are made up of the normal character set (`'a'`,...,`'z'`) along with the character `'-"` which is used in some hyphenated words e.g. `copper-colored`.
- The words in the story files may contain capital letters (such as at the start of a sentence).
- As well as containing words, the story files may contain whitespace (spaces, tabs, new lines) and punctuation marks.
- All whitespace characters should be preserved in the translated file *exactly as they are* in the original file.
- Words that begin with a capital letter in the original file should begin with a capital letter in the translated file. Except for possibly the first letter, translated words should be lowercase in the translated file. So, if the English word `ComPlaint` appears in the original story, the Elvish word `Nur` should be output to the translated story.
- Punctuation marks (`, .  '` etc) should be preserved in the translated file exactly as they appear in the original file except for the special case of the star (`'*'`) character.
- Words that appear between two stars e.g. `*unusualWord*` are assumed to be *already translated* and are output *exactly* as they appear in the original file with the stars removed. In this case, `unusualWord` should be output.
- It may be the case that there are some words in the file that are not contained in the dictionary. These words should be output *exactly* as they appear in the original file, but surrounded by two stars. For example, the English word `around` does not appear in the dictionary. Therefore, when translating into Elvish, if `around` occurs in the story, it should be output as `*around*` in the translated story.

- Stars will only appear in the input stories surrounding words that should not be translated. Your program may assume that stars only appear in the story in this way.

While this may seem a bit daunting, the assignment uses input files of increasing difficulty. If you get your program to work for even the simplest input file, *this will be sufficient to pass the assignment.* The input files are described as follows:

- `simpleStory1.txt`: This story consists of a set of lowercase English words, separated by whitespace. All the English words are in the dictionary and therefore there is a corresponding Elvish word for each one. It does *not* contain any punctuation marks. You should translate each word, but also preserve the white space exactly as it appears in the story.
- `simpleStory2.txt`: All the English words in this story are also in the dictionary. However, some of the words begin with capital letters and there are punctuation marks (',' and '.') which need to be preserved in the translated story.
- `story1.txt`: This story contains lots of words that are not in the provided dictionary, as well as words that do appear in the dictionary. All words that do not appear in the dictionary should be surrounded by stars in the translated story.
- `huge.txt`: The story is similar to `story1.txt` but is much larger. It can be used to gauge the efficiency of your program.

Although I am providing these four example input files, I will use four different files to test your code on.

Marking Scheme

Pay attention to the marking scheme, as you can obtain a high mark without necessarily solving all parts of the problem. The programs will be marked automatically, based on how they execute. I will not examine the source code. This is how marks are assigned:

- The program successfully compiles. (20 marks).
- The program runs on a test file similar to `simpleStory1.txt` and generates a correct `story_in_elvish.txt` file. (20 marks)
- The program runs on a test file similar to `simpleStory1.txt` and generates a correct `story_backto_english.txt` file. (20 marks)
- The program runs on a test file similar to `simpleStory2.txt` and generates a correct `story_in_elvish.txt` file. (5 marks)

- The program runs on a test file similar to `simpleStory2.txt` and generates a correct `story_backto_english.txt` file. (5 marks)
- The program runs on a test file similar to `story1.txt` and generates a correct `story_in_elvish.txt` file. (10 marks)
- The program runs on a test file equvialent to `story1.txt`. The `story_backto_english.txt` file only differs from `story1.txt` where there is ambiguity in the translation. (10 marks)
- The final 10 marks are determined based on the *speed* of the program when run on a test file similar to `huge.txt`.