CS 253: Problem Solving with C++ (Fall 2006)	**Assignment 6**
Instructor: Dr. Indrakshi Ray	**Spelling**
TA: Kevin Depue	**Due 12/6/2006**

1 Introduction

In this assignment, you will be creating a spell checker using two different underlying data structures: a list and a hash table. In the past I have talked briefly about asymptotic complexity. Many of you do not quite understand this concept. This assignment will hopefully make it more clear. We will see that spell checking a large document using a list data structure will take several minutes whereas using a hash table data structure takes a matter of seconds!

2 Command Line Requirements

When your code is compiled, it should produce an executable called `spelling`. The syntax for running your program via command line should be the following:

```
spelling <mode [-l == list,-h == hash table, -b == both]>
         <printing [-p == print misspelled, -np == don't print misspelled]>
         <dict_file> <input_file>
```

Here, `mode` tells us that we're either spell checking using a list data structure, a hash table data structure, or both. `printing` simply refers to whether or not we print out the list of misspelled words when the program is run. The `dict_file` parameter is a text file containing the dictionary to be used, and the `input_file` is the file to be spell checked.

You should set up your code so that if I type `spelling` without any command line arguments, it prints out the command line syntax:

```
Syntax: spelling <mode [-l == list,-h == hash table, -b == both]>
                 <printing [-p == print misspelled, -np == don't print misspelled]>
                 <dict_file> <input_file>
```

Here, the `>` is just the command line prompt. If you do not print out the syntax when I type spelling, I will take off points. You will find this coding paradigm helpful when you forget what the command line arguments to your old programs are. **Since this is the last assignment I've already done this for you.**

As an example, here is the command for spelling checking a file called `test.txt` using a dictionary `dict.txt` using both the list and hash table data structures and printing out all misspelled words:

```
spelling -b -p dict.txt test.txt
```

3 Program Requirements

This assignment should be very straightforward. Your program should first read in the dictionary file specified on the command line. When reading in the dictionary, store an initial copy as an STL `vector` of STL `string`'s. Also, make sure that you call `ToLower` on every word in the dictionary. I have already written this method for you; it resides in `Dictionary.h`.

Now, read in the input file to be spell checked specified on the command line. When reading in each word, make sure you make a call to `TrimPunctuation`. This method resides in `Dictionary.h`, and trims all punctuation characters from the *ends* of the word. If you trim punctuation on a word and there are no alphanumeric characters left, you should not spell check it. You should implement the `TrimPunctuation` method. A major hint here is to use the built-in function `ispunct`. This method returns true if the character passed in is a punctuation character. You should test your function to make sure it works. Following are some examples:

`TrimPunctuation("?.Hello,*")` → "`Hello`"
`TrimPunctuation("?.,*")` → ""
`TrimPunctuation("?.b,*")` → "`b`"

Now, if we are to spell check using a list data structure, you need to call `LoadLinkedListDictionary`. This loads the dictionary into an STL `list` via reference. Then, you need to make a call to `ListSpellCheck`. Note that this method will add all misspelled words to an STL `vector` via reference. There is a method `IsMisspelled` that returns true if a given word `w` is not found in the list-based dictionary to make writing the `ListSpellCheck` method easier. Note that for each word in the input file, we must compare it with *every other* word in the dictionary. This gives us a time bound of $O(n^2)$. Also, note that although we want to print (if specified via command line) the misspelled words as they actually appear in the input file, we must convert each word to lowercase before checking if it resides in the dictionary.

You must implement the hash table spell checking methods as well. These are very similar to those in list, except that `IsMisspelled` for hash tables is much simpler. I have already set up the hash table data structure and written the hashing function for you. Refer to the STL online documentation for `hash_set` to learn how to insert things into the data structure and how to determine whether something resides in the data structure. Searching for a word in the hash-based dictionary takes constant time, giving us a time bound of $O(n)$. This is much better than the $O(n^2)$ bound we got using the list data structure.

To show you how much slower growing $O(n)$ is than $O(n^2)$, I have provided you with a simple `Timer` class so that you can determine how long each spell checker takes in seconds. You should only time the actual spell checking routines (e.g., `ListSpellCheck` and `HashSpellCheck`). Here's what your output should look like:

Command:

```
spelling -b -p dict.txt small.txt
```

Input (`small.txt`):

```
Hlllo there mate!  How aer you diong today? That's great mna.  Good for yoo.
```

Output:

```
Misspelled
----------
Hlllo
aer
diong
mna
yoo

Input Size:      14
Dictionary Size: 175171
Num Misspelled:  5,5
List Time:       0.089019 seconds
Hash Time:       2.30001e-05 seconds
```

Note that all statistics are aligned at the left. If you don't do this, I will take off points. Also, you only print out the misspelled words if the command line option `printing` is equal to `-p`. When printing out the number of misspelled words, you should print out the number of misspelled words found by `ListSpellCheck` *and* the number of misspelled words found by `HashSpellCheck`, separated by a comma. This will tell you if both data structures are giving you the same results. If only one or the other is to be run (e.g., the `mode` command line option is either `-l` or `-h`), just print out the number of misspelled words for that data structure. Finally, print out the running time for the list data structure, the hash data structure, or both, depending on the `mode` command line option. For example, if I only spell check using the list data structure and don't print misspelled words, my output would look like this:

Command:

```
spelling -l -np dict.txt small.txt
```

Input (`small.txt`):

```
Hlllo there mate!  How aer you diong today? That's great mna.  Good for yoo.
```

Output:

```
Input Size:      14
Dictionary Size: 175171
Num Misspelled:  5
List Time:       0.089051 seconds
```

On a final note, you should implement the overloaded `<<` operator at the top of `main.cpp` to make it easier to print out STL `vector`'s of STL `string`'s.

I will provide you with a good text-based dictionary. It will be on the web, and was originally obtained from [1]. In addition, you may want to test your program on some text-based books. I will provide you with one on the web (`carol.txt`), and you can find several other free text-based eBooks here: [2]. Try running your program on `carol.txt` using both the list and hash table data structures. How do the running times compare?

4 Submission Instructions

When you are ready to turn in your assignment, you will need to do the following:

- Make sure your code and makefile reside in a directory named `assign6`.
- tar up *the directory* containing your code and makefile. *Make sure* you include the directory. If you do not, I will take off points. The file you submit should be named `assign6.tgz`. If it is not named `assign6.tgz` I will take off points. Create this tar file by typing the following command on a department Linux machine:

  ```
  tar -czvf assign6.tgz assign6/
  ```

- Submit your assignment by running the following command on one of the department Linux machines:

  ```
  cs253/bin/checkin assign6 assign6.tgz
  ```

  You can resubmit your assignment as many times as you like before it is due by running the above command. More information about the checkin program can be found here: `http://www.cs.colostate.edu/~info/checkin_prog.html`.

When I grade your assignment, I will type the following commands on a department Linux machine:

```
tar -xvf assign6.tgz
cd assign6/
make
```

The resulting executable should be named `spelling`. If it is not named `spelling`, I will take off points. You should run the above commands on a department Linux machine to make sure that your tar file does what I expect so that you will not lose points. Part of your grade is to follow directions correctly.

5 Grading

The grading will be based on two different things. There are many places in this document where I say, "Do this, or I will take off points". If you follow directions, you should be fine and receive all those points. Additionally, I will be running your program on two examples that you have never seen before; a small example and a very large example.

6 Due Date

This assignment is due on 12/6/2006. No late homework will be accepted unless you have talked with the instructor/TA in advance.

7 Acknowledgments

I'd like to thank Andy Curtis for coming up with this assignment idea.

References

[1] The biggest viewable dictionary on the 'net. http://www.orchy.com/dictionary/index.html.

[2] Project gutenberg. http://www.gutenberg.org/wiki/Main_Page.