EECS 280 Lab 05: Strings and IO

Due Sunday, 14 February 2016, 8:00pm

In this lab, you will practice string manipulation (using both C-strings and `string` objects from the C++ Standard Library) and file input/output in C/C++ and use them to implement a simple spell checker.

This lab covers material from these lectures:

- 06 Arrays and Pointers
- 07 Array Traversal
- 08 Argv, IO, Enums and Templates

Overview

Completion Criteria:

- ✓ (Task 1) Implement `strcmp_eecs280`. The code must be reasonably close to correct..
- ✓ (Task 2) Implement `getUserWord` and `findWord`. The code must be reasonably close to correct.

Task 0 - Preliminaries

The Files

We have provided starter files for this lab. If you have a terminal open, the following command will automatically download it from the eecs280 Google Drive repository to your current directory:

```
$ wget goo.gl/gyIKaa -O - | tar xzk
```

In case you are working locally and want to manually download the files, they are also attached to the CTools assignment and available on the course Google Drive Repository (see link in header).

Here's a brief summary of the files included in this lab. Files you need to turn in are shown with a red background.

`lab05.cpp`	Contains function stubs for `getUserWord` and `findWord`. This file includes the `main` function and testing code.
`words.txt`	A dictionary of properly spelled words, one word per line.

Testing Code

`lab05.cpp` contains a `main` function with testing code we've written for you. Compile it with:

```
g++ -Wall -Werror -O1 -pedantic lab05.cpp -o lab05
```

The starter code should "work" out of the box, so make sure you are able to compile and run it. The code may be missing some pieces, contain some bugs, or crash when you run it, but you'll fix each throughout the course of the lab.

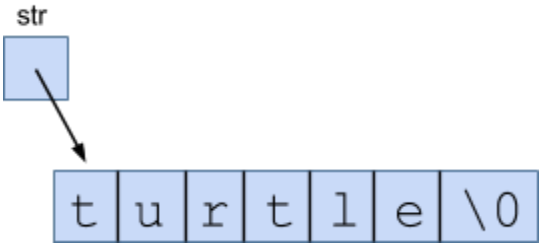
Introduction

The overall goal of this lab is to learn about string manipulation and file input/output in C++, and to do this we'll work through the implementation of a simple spell checker. In order to determine if a word is spelled correctly, we will search through a text file containing an exhaustive list of words and look for an exact match. If one is found, we assume the word was spelled correctly. Otherwise, we assume the entered word is a misspelling.

Note: Here and throughout the lab, we assume all words are written using lowercase characters. This is true for the reference file and you should only input words in lowercase when running the program.

Task 1 - Checking if Two Words Match

Let us first consider how we could solve this problem by using C-strings to represent words. Recall that a C-string is simply an array of characters that terminates with a null character, `'\0'`. We don't have to use an extra variable to keep track of a C-string's length because we can use the null character as a *sentinel* for iteration. In the diagram below, `str` is a pointer to the first char in an array containing the C-string `"turtle"`.



So it seems clear we can determine if two words are the same by iterating through the C-string representing each and comparing characters one by one. Let's go a little bit farther and actually replicate the `strcmp` function from the `<cstring>` library, which determines how two C-strings are ordered. Find the `strcmp_eecs280` function in your `lab05.cpp` file and write an implementation according to the RME.

```
// REQUIRES: str1 and str2 point to C-strings
// EFFECTS:  If st1 and str2 are identical (contain exactly
//           the same characters), returns 0.
//           If the first differing character has a greater
//           value in st1 than in str2, return a positive number.
//           Otherwise, return a negative number.
int strcmp_eecs280(const char *str1, const char *str2){

  return 0;// TASK 1 - REPLACE WITH YOUR CODE

}
```

For example:

`strcmp_eecs280("turtle", "frog")` should be positive
`strcmp_eecs280("turtle", "turtles")` should be negative
`strcmp_eecs280("", "frog") =` should be negative
`strcmp_eecs280("lizard", "lizard") = 0`

Take note that `strcmp_eecs280` (and the original strcmp from the `<cstring>` library) never make any promises about what positive or negative number they return. Thus, you should always check something like `strcmp("turtle", "frog") < 0` rather than `strcmp("turtle", "frog") == -1`.

Task 2 - Checking Against the Words File

Now that we have a way to compare words, we need get a word from the user and compare it to the reference file `words.txt`. We want to prompt the user to enter a word, but because we don't know how long the word is ahead of time (i.e. before we run the program), using C-strings would be difficult (*why?*). Instead, we'll work with `string` objects from the C++ Standard Library that can hold words of arbitrary sizes.

Complete the **`getUserWord`** function according to its RME. The initial implementation just returns `"quit"` because that will cause the testing code to stop right away.

```
// EFFECTS:  Prompts the user to end a word using the prompt
//           "Please enter a word:  " and then reads a string
//           from cin which will be returned. After reading the
//           input, cleanup by printing a newline to cout.
string getUserWord(){

  return "quit"; // TASK 2 - REPLACE WITH YOUR CODE

}
```

Once we have the user's word on hand, we need to compare it against each word in the `words.txt` reference file. We'll encapsulate this in the **`findWord`** function. Write this function according to the RME given. You MUST use the strcmp_eecs280 function you wrote in part 1.

```
// EFFECTS:  Searches words.txt for the word passed as a
//           parameter.  If found, return true.  Otherwise false.
//           If words.txt cannot be opened, prints an error
//           message to cout and returns false.
// NOTE:     You MUST use the strcmp_eecs280 function.
bool findWord(string word){
  return false; / TASK 2 - REPLACE WITH YOUR CODE
}
```