Lab 3
Strings

15 September 2015

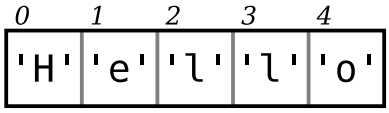
The drill for this lab is given below, after a short reading on how strings work and some of the operations we can do on strings. Come to lab on Tuesday either with it completed or with a specific written question in your notebook identifying which drill step you got to and what about it you're stuck on.

String info

There's not really any single good place in our textbook that covers string internals (at least, not that would be accessible this early in the term), so I want to present some of them here:

- Strings are made up of characters; each element of a particular `string` value is itself a `char` value.
- The position of each character in the string is called its "index" within the string. Some people use "indexes" as the plural of "index" and others use "indices"; you'll probably hear me use both.
- An index represents how far past the first character you need to go, so the first character is actually at index 0 (because if you're at the first character you don't need to go any further). The character at index 1, one past the first, is the second element of the string, and so on.
- The length of the string is the actual number of characters in it; the last index is thus always one less than the length.

Here's an illustration of the five-character string `"Hello"`, with indices marked:



And, a selected list of things you can do to or with strings, a couple of which you've already seen and several that are new. All the expressions in the table below assume you have a `string` variable named `s` that's already got a value, but they'll work with *any* variable that's a `string` (hopefully, in actual code, with a more descriptive name than `s`).

`s = "foo"`	assigns the value `"foo"` to be the new contents of the string variable—works with any `string` after the equal sign
`s == "foo"`	determines whether the string is the same as the value `"foo"`—works with any `string` after the double-equal sign
`s + "foo"`	computes the string that would result from concatenating the string `s` with the string `"foo"`—works with any `string` after the plus sign
`s[2]`	retrieves the character at index 2 (that is, the third character)—works with any non-negative `int` expression in the square brackets, as long as it's not past the end of the string
`s.front()`	retrieves the first character in the string
`s.back()`	retrieves the last character in the string
`s.length()`	retrieves the number of characters in the string
`s.substr(2,3)`	computes the portion of the string that starts with the character at index 2 and continues for a total of 3 characters—works with any non-negative `int` expressions instead of 2 and 3
`s.find('x')`	computes the index of the first occurrence of the character `'x'` in the string—works with any `char` expression inside the parentheses. If the given character is not in the string, returns the special number `string::npos` (so you can ask if the `s.find('x') == string::npos` and if it does, then `'x'` was not found in the string)

String drill

There's no drill in the book on this topic, but this part of the lab is philosophically similar to an end-of-chapter drill: it's somewhat contrived but lets you write a short program to practice the basics.

Make your directory for this lab, and start its its `README.TXT` file. (From here on out, I'm not going to give you a checklist of what-all to include in your readme, because you have done a few and know what belongs in them, but I do always expect you to have one, and it should reflect the actual instructions and documentation for how to work with your code.)

The first program in this week's lab will be written in a file called `stringinfo.cpp` and it will read a single string—specifically, it will read one string that doesn't have spaces in it—and print out information about it.

1. First, write the program to read the string and print out the message

   ```
   The string "______" has _ characters.
   ```

   except with the blanks filled in correctly for that string: the string itself and its length (how many characters it has). Note that to include a double-quote inside a double-quoted string, you have to use `\"` for the quote inside the string (as with `\n`, the backslash indicates that something special is happening).

2. Add second line to the output that says

   ```
   Its first is '_' and its last is '_'.
   ```

   filling in the blanks with the first and last characters of the string (you can assume it will not be empty).

3. In cases where the string is at least three characters long, add a third line

   ```
   If you trim the first and last characters it leaves "____".
   ```

   The blank here should have the whole string *except* for its first and last characters.

4. Finally, in cases where there is a hyphen in the string, make it print a last line

   ```
   It has a hyphen at index _.
   ```