CISC 3142
Programming Paradigms in C++
Lab #6
C-Strings

How to Develop and Submit your Labs

Lab 6 — C-Strings

Overview

This topic was actually a lecture topic in 3110 (i.e., the second semester programming course when C++ was the primary language of instruction in the department), however, since you're all seasoned programmers by now, we're going to make it an assignment.

First, here is a writeup on C strings, i.e., strings as represented in C (and often used as the underlying structure for the C++ `string` class).

In the Standard C Library there is a library (i.e., header) named `cstring`, containing a collection of functions used to manipulate C-Strings. In this exercise, you are to implement several of the functions provided by the `cstring` library. Since we are 'hijacking' the C-String library, there is concern about name clashing — we will use a namespace to avoid that — see below. (In actuality, the clashes don't really arise until the upcoming related lab that has you define a simple string class; but we'll get started with the namespace now, while things are simple.)

The Details

Implement a `cstring` module (with a `.h`/`.cpp` pair) containing the following C-style (i.e., non-member) functions:

```
char *strcpy(char *dest, const char *src);
char *strcat(char *dest, const char *src);
int strlen(const char *str);
```

the module should belong to the namespace `mystring` (see below). (If you really want some fun, try coding a couple of the above **recursively** — `strlen` is a good place to start).

I've provided a `cstring_app.cpp` test driver.

Namespaces

Here is a more detailed writeup on namespaces.. For the purposes of this lab, the following should be sufficient.

Defining the Namespace

To avoid the possibility of name clashing, we will place our C-string library in a namespace named `mystring`. The contents of a namespace may be distributed across multiple files — source as well as header. The basic syntax for placing code within a namespace is:

```
namespace <namespace-name {
        …
        // code belonging to the namespace
        …
}
```

Pragmatically speaking, in a header (`.h`) file, place this between the include guards (i.e., have the include guards be the outermost code of the file, then enclose everything else, e.g., the class declaration, within the namespace body. For a source file, have all the includes and using appear before the namespace, and the rest of the file (i.e., your actual code) be enclosed within the namespace body as well.

Using Your Namespace

So now you have two namespaces to deal with — `std` and `mystring` (there's actually a third you've been using, the global namespace). Recall that when using an entity (type, object, function, etc) from a namespace you must provide additional information to help the compiler find the entity. This can be done by either:

- supplying the fully qualified name (i.e., including the namespace name) with the scope resolution operator, such as `std::cout` or `mystring::strcmp`
- using a using directive, e.g. `using namespace mystring`; this allows the entire contents of the specified namespace to be used without requiring the scope resolution operator.
    - You can have several of these active at the same time. Remember however, that referring to an entity whose name appears in both namespaces will cause an ambiguity error
- using a using declaration, e.g. `using std::cout`; this brings the specified entity into the current scope and allows it to be used without requiring the scope resolution operator.
    - Again, you can have more than one of these, but again clashes can occur.

The primary purpose of the using directive and declarations is to allow specification of an entity by its unqualified (i.e., simple) name. When multiple namespaces are in play — especially when there is an issue of possible name clashing, the question becomes whether to use a directive or declaration (there are also schools of thought that insist on one or the other's use, regardless of the specific situation). What seems to usually work is to use the directive for the namespace which contains the bulk of entities you want to work with, and either qualify or use a using declaration for any others.

- Fun with pointers and C-strings
    - This will give you a chance to practice basic pointer operations and arithmetic in a natural and practical situation
- You will use your C string class as the underlying structure for your own `string` class later on
- A natural use of namespaces.

Code Provided for this Lab