ITP 11706 Project #3 – String Library Spring 2011

Due date: May 23, 20:00

1 Purpose

This project will give you plenty of practice using pointers, and will illustrate the concept of a "library".

2 Project description

Programmers frequently make use of "libraries" to extend the language they are using to handle some specific set of tasks. For example, we use the library `iostream` to enable us to read and write input and output stream in C++ language.

For this project, you will write your own string library, which could take the place of the standard `string` library. As you are creating your library of functions you are **NOT** allowed to call any of the functions that are provided in the standard `string` library.

As you are implementing this project, three files will be provided in /class/itp1106/P3/

```
Makefile         [A makefile that helps you compile multiple source codes]
mystringlib.h    [A library header file]
driver.cpp       [A small program that will test your string functions]
```

With the provided three files, you should write

```
mystringlib.cpp  [The library source code]
```

2.1 mystringlib.cpp

`mystringlib.cpp` will contain the definitions for functions with the prototypes listed below. **This file must contain only the definitions for the functions below, and nothing else. In particular, this file must NOT contain the function "main".** You may include one or more "helper" functions if you wish. (A helper function is a small function that is called by one or more of the functions in your library to perform a specific task. A helper function is **not** considered part of the library; it is not intended to be called outside of the library itself.) Please note that the prototypes below are not exactly the same as the prototypes for the corresponding functions that we find in `string.h`. I have made small changes to simplify your task. Here are prototypes for the functions that `mystringlib.h` defines `mystringlib.h`:

```
int mystrlen(const char *);    [Note: the argument type is const char*, not string]
char *mystrcpy(char *, const char *);
char *mystrncpy(char *, const char *, int);
char *mystrcat(char *, const char *, int *);
int mystrcmp(const char *, const char *);
char *mystrchr(char *, char);
char *mystrstr(char *, const char *);
char *replace(char *, const char *, const char *);
```

Your functions should behave precisely like the corresponding functions that are part of the library `string.h` (with a couple of minor exceptions, described in the section below.)

For example, your implementation of `mystrlen` should work exactly like the function `strlen`. For information about how these functions behave, please refer to manual page by running `man function_name` in Linux shell. Again, I have modified the prototypes very slightly to make it easier for you; you must use the prototypes as they appear above.

The string functions in the standard library do not do any error checking of any kind. In particular, it is up to the person who uses these functions to ensure that there is adequate space available for the string manipulations. Similarly, your functions will just assume that anyone who is using your library will be careful to allocate sufficient space in memory. Your functions should not do any error checking of any kind.

To avoid repetition, some of your functions may call other functions that you have already written. However, **you may NOT use any of the functions that are declared in string.h at any time!** Also, **you are NOT**

allowed to use string type variables. If you use any **string** objects in the library, all your outputs will be considered incorrect. If I allowed you to call functions like `strlen` and `strcpy` then this project would be much too easy.

To ensure that you get plenty of practice using pointers, **you may NOT use square brackets ('[' and ']') at any time while writing your function library.** You may want to use the square brackets while modifying `driver.cpp`, however. Again, all your outputs will be considered incorrect if you use [] in mystringlib.cpp.

Note: For some library functions, you would have to call **realloc()** to resize the memory space assigned for a pointer instead of using new[] and delete[].

2.1.1 Exceptions

- `mystrcat` has a third parameter. The third argument specified by the user will be the address of an integer variable. The function will initialize this variable with the number of characters that were concatenated onto the original string (do not count the NULL character).
- `mystrcmp` will return 0 if the two parameters are identical, -1 if the first parameter is "less than" the second parameter, and 1 if the first parameter is "greater than" the second parameter.
- There is no "replace" function in the standard string library so I must describe its implementation. The three parameters all represent strings. For this discussion, assume that `a`, `b`, and `c` are char pointers, and that we call the function like this: `replace(a, b, c);` The function will search string `a` for the first occurance of the string `b`. If an occurance is found, it must be removed and replaced with `c`, and the function will return a pointer to the position in the string `a` where the first character of the substitution occurs. If an occurance is not found, the function will return the NULL pointer. As an example, if `a` points to the string "hello there Fred", `b` points to the string "there", and `c` points to the string "magnanimous", then after the function call, `a` would point to the string "hello magnanimous Fred". (Note that the string represented by the first argument could become longer or shorter as a result of the function call.)
- You may wonder why the first parameters in the functions `mystrchr` and `mystrstr` are not `const`. In the real world they should be, but for this project it would introduce an undesireable complication, so please follow instructions and do **not** make them `const`.

2.2 driver.cpp

The file `driver.cpp` will contain the function `main` (plus any other functions that you find useful.) The purpose of this file is to make various calls to the functions in your string library to test them as thoroughly as possible. How you choose to write this file is entirely up to you! Keep in mind that any functions included in this file are for your own testing purposes only, and should not be called by any of the functions in your string library. You should continually make changes to this file so that it will carefully test the portion of the string library that you are currently working on. So that you can call the functions in your library from inside the file `driver.cpp`, the first line in the file `driver.cpp` should be:

```
#include "mystringlib.h"
```

(Note that we use quotes instead of angled brackets to include a file that we have written ourselves.) Now to test your library with the driver that you have written, all you have to do is type:

```
$ g++ -g -Wno-write-strings driver.cpp mystringlib.cpp
```

or if you have copied the Makefile to your current directory, simply type:

```
$ make
```

Note that **you will only submit the file `mystringlib.cpp`.** We will use our own drivers to test your library very thoroughly, so please do not try to submit your driver.

2.2.1 An example of a very simple driver

Suppose that you have only written the function `mystrlen` and `mystrcpy`, and you want to develop a driver to test it. The following is a simple example to get you started. (Note that it does not do a very thorough job of testing `mystrlen`!)

```
#include "mystringlib.h"
#include <iostream>
using namespace std;

int main()
{
  char *a = new char[7];
  mystrcpy(a, "Testing");

  cout << mystrlen(a) << endl;
  cout << a << endl;

  delete[] a;
  return 0;
}
```

If you compile and run this small driver, the executable should display the value 7 and "Testing". If it doesn't, then you know there is something wrong with your implementation of the function `mystrlen`.

3 The "primary case"

Our method of grading this project will be very different from what we have done for project 1 and 2. We will test your function library with a variety of different drivers that will make calls to your functions. As usual, we want to provide you with an opportunity to test your project with the "primary case" so that you can check your output. Instead of providing a `primary_input` file as we have done in the past, we are giving you a file called `driver.cpp`, `mystringlib.h` and `Makefile` in /class/itp117/P3/.

`make` command will generate an executable, `a.out`. If you run `a.out`, it should produce output that matches the file `primary_output.txt`. Notice that there is no input file required – the driver has the primary input hard-coded inside. To use our driver, you must have finished writing all of the functions that are supposed to be included in your library. I don't recommend using the provided driver during the development process. As you are writing your library of functions, you should be continually re-writing your own driver(s) to test the specific component that you are working on at the time.

IMPORTANT: You are only expected to submit a working library of string functions, called `mystringlib.cpp`. When your library is linked with our driver (as described above), the resulting executable should produce the output you see below. This output shows the results of several succesive calls to the functions in your library. There are two strings, `a` and `b`, which are manipulated during the program execution. After each succesive function call, the new values of `a` and `b` are displayed, along with the return value. Here are the contents of the file `primary_output`:

```
mystrcpy(a,"pizza")
return value: pizza
value of a: pizza
value of b: Unknown

mystrcpy(b,"taco")
return value: taco
value of a: pizza
value of b: taco

mystrlen(a)
return value: 5
value of a: pizza
value of b: taco

mystrcpy(b,a)
return value: pizza
value of a: pizza
```

```
value of b: pizza

mystrcat(a,b,&c)
return value: pizzapizza
value of a: pizzapizza
value of b: pizza
value of c: 5

mystrcmp("doghouse","dog")
return value: 1
value of a: pizzapizza
value of b: pizza

mystrcmp("Fred","Fritz")
return value: -1
value of a: pizzapizza
value of b: pizza

mystrcmp("Tony","Tony")
return value: 0
value of a: pizzapizza
value of b: pizza

mystrchr("abcdefghijklmn",'g')
return value: ghijklmn
value of a: pizzapizza
value of b: pizza

mystrchr("abcdefghijklmn",'x')
return value: NULL
value of a: pizzapizza
value of b: pizza

mystrncpy(b,"dog",7)
return value: dog
value of a: pizzapizza
value of b: dog

mystrncpy(b,"hello there",2)
return value: heg
value of a: pizzapizza
value of b: heg

mystrstr("how are you today?","you")
return value: you today?
value of a: pizzapizza
value of b: heg

replace("how are you today?","you","me")
return value: how are me today?
value of a: pizzapizza
value of b: heg

replace("how are me today?","are","was")
return value: how was me today?
value of a: pizzapizza
value of b: heg
```

4 Submitting your project

Turn in your program using the "submit" program as before, except using "proj3" for the project number. You are to submit only the mystringlib.cpp file containing your source code, not the executable version of your program! If your program is in a file named mystringlib.cpp, submit would be run as shown.

```
$ submit itp11706 proj3 mystringlib.cpp
```