1 Problem Description

We have introduced and studied the concept of pointer variable and learned that the name of an array is a constant pointer to the type of object stored in the array. Array name, as a pointer, stores the address of the first element of the array. C-style string is an array of char that is terminated by a null character, that is, '\0' (the value could be implementation dependent, but most implementations use all 0 in each bit). To practice our understanding of pointers and arrays, as well as our programming skills, in this lab we are going to implement some standard functions in the C/C++ library that manipulate C-style strings. In general, C programming language may be viewed as a subset of C++ programming language. In C programming language, string type or class (C++ string) is not available. So to deal with strings of characters, arrays of characters are used. In fact, the implementation of C++ string uses C-style string and this could be another motivation for this lab. We will study how to implement C++ string in our other labs and a programming project.

In this lab, we will first use three functions, strlen, strcat, and strtok, in the standard library in our program. After that we will implement two of the three functions: strlen and strcat. The implementation of strtok is the task of our first assignment.

Typing "man strlen" on the prompt in a linux machine such as turing, we get the following. Try to read the documentation to see if it makes sense to you. At this point, it is normal if the document does not seem to make sense to us. We hope that the document will make more sense to us as we move forward in developing our programming and computational skills and understanding.

```
wingning@turing:~/csce3143$ man strlen
STREN(3) Linux Programmers Manual STREN(3)

NAME
  strlen - calculate the length of a string

SYNOPSIS
  #include <string.h>

  size_t strlen(const char *s);

DESCRIPTION
  The strlen() function calculates the length of the string s, not including the terminating \0 character.

RETURN VALUE
  The strlen() function returns the number of characters in s.

CONFORMING TO
```
Please pay attention to the function prototype. To describe the above in another way, the parameter or argument to the \texttt{strlen} function is pointer to char. This pointer "should" points to a C-style string, meaning the end of the string is indicated by '\0' the null character in one of the cell of the array, which is assumed by the function. Therefore, any user of this function must make sure that the assumption is met or serious error may result. The function returns the number of characters between the location pointed to by the pointer parameter and the ending null character.

Typing "man strcat" on the prompt in a Linux machine such as turing, we get the following. Try to read the documentation to see if it makes sense to you again.

```bash
wingning@turing:~/csce3143$ man strcat
```

**NAME**

\texttt{strcat}, \texttt{strncat} - concatenate two strings

**SYNOPSIS**

```c
#include <string.h>

char *\texttt{strcat}(char *\texttt{dest}, const char *\texttt{src});
char *\texttt{strncat}(char *\texttt{dest}, const char *\texttt{src}, size_t \texttt{n});
```

**DESCRIPTION**

The \texttt{strcat()} function appends the \texttt{src} string to the \texttt{dest} string, overwriting the null byte (\texttt{\0}) at the end of \texttt{dest}, and then adds a terminating null byte. The strings may not overlap, and the \texttt{dest} string must have enough space for the result.

The \texttt{strcat()} function is similar, except that

* it will use at most \texttt{n} characters from \texttt{src}; and
* \texttt{src} does not need to be null terminated if it contains \texttt{n} or more characters.

As with \texttt{strcat()}, the resulting string in \texttt{dest} is always null terminated.

If \texttt{src} contains \texttt{n} or more characters, \texttt{strcat()} writes \texttt{n+1} characters to \texttt{dest} (\texttt{n} from \texttt{src} plus the terminating null byte). Therefore, the size of \texttt{dest} must be at least \texttt{strlen(dest)+n+1}.
A simple implementation of `strncat()` might be:

```c
char*
strncat(char *dest, const char *src, size_t n)
{
    size_t dest_len = strlen(dest);
    size_t i;

    for (i = 0 ; i < n && src[i] != \0 ; i++)
        dest[dest_len + i] = src[i];
    dest[dest_len + i] = \0;
    return dest;
}
```

**RETURN VALUE**
The `strcat()` and `strncat()` functions return a pointer to the resulting string `dest`.

**CONFORMING TO**
SVr4, 4.3BSD, C89, C99.

**SEE ALSO**
bcopy(3), memccpy(3), memcpy(3), strcpy(3), strncpy(3), wcscat(3), wcsncat(3)

**COLOPHON**
This page is part of release 2.77 of the Linux man-pages project. A description of the project, and information about reporting bugs, can be found at http://www.kernel.org/doc/man-pages/

Please again pay attention to the function prototype. To describe the above in another way, there are two parameters or arguments to the `strcat` function. Both are pointer to char. The pointers "should" point to a C-style string, meaning the end of the string is indicated by '\0' the null character in an array cell, which is assumed by the function. Also, there must be sufficient space or memory after the terminating null character in the first argument, so that characters in the C-style string of the second argument can be stored or copied over. Therefore, any user of this function must make sure that the assumption is met or serious error may result. The function returns the first argument, which points to result of the concatenation of the two C-style strings.

## 2 Purpose

Understand pointer and array, and in particular pointer to char, array of char, and C-style string. Practice writing functions that are similar to standard library functions to show our ability and command of basic data structures and programming in C/C++ and to build our confidence. Learn a few library functions that manipulate c-style strings.
3 Design

For this part, we will follow what is provided in the documentation except for the names. We could use the identical names as the standard library functions if we create our own namespace, otherwise name conflict or clash may result. We will prefix the name with my.

\[
\text{size_t mystrlen(const char *s);}\
\]

\[
\text{char *mystrcat(char *dest, const char *src);}\
\]

Do not be concerned about what \text{size_t} means at this time. It will be taken care of with the header or include files.

4 Implementation

There are three subtasks in the following. First we write a simple main program to use the three functions in main.cpp file. The details of what the main program should do are given below. Next implement the mystrlen function in mystrlen.cpp file. Then implement the mystrcat function in mystrcat.cpp file. We will have three source files.

1. We will write a simple program that simply uses the three standard library functions. The following is a high level description of the main program. In this program, we declare and define the following variables and carry out the following actions:

(a) two size_t type variable called \text{size} and \text{len},
(b) an array of 10 elements where each element has the type of pointer to char called \text{strArray},
(c) a C-style string defined by \text{char line[] = "ls -l -a | wc -c > myfile"},
(d) a pointer to char variable called \text{result},
(e) use \text{strlen} function to compute the length of C-style string \text{line} and store it in \text{len}; output the value of \text{len} (The length of line is...)
(f) use \text{strtok} function and space as a delimiter to break the C-style string \text{line} into pieces (the first piece is ls and we have a pointer points to it, the second piece is -l, ...) and store the pointer to each piece (the return value of strtok) in \text{strArray}. Even though we may “hard code” a solution here as we know the value of line, we should use a loop and testing the return value of strtok to stop the loop; output the values of \text{strArray} array one per line( \text{strArray[0]} = ls...) 
(g) compute the sum of string lengths of strings in \text{strArray} using a loop and store it in \text{size},
(h) compute the number of blanks (or spaces) in `line` (should be easy to compute, can you see it?) and output the value (The number of blanks is...),

(i) What do you think the value of `line` is at this point? Write down your answer and check your answer with the program by adding output statement.

(j) use `new` to allocate an array of `size+1` characters and store the address of the first element (or the beginning address) in `result`,

(k) use `strcat` to concatenate all C-style strings in `strArray` into `result`. Use a loop and set the first entry of `result` to ‘\0’ to achieve it (`new` may already have initialized it for us, but making it explicit helps us to see conceptually what is needed). Output the value of `result` (The concatenation of all strings in `strArray` is ...)

2. Implement `mystrlen` function. Here is the idea: initially the parameter `s` points to the first element. You may either use array notation `s[index]` or `*s` notation (try to use both, one for `mystrlen` implementation and the other for `mystrcat` implementation, so you really understand the concepts about pointer and array) to dereference the pointer so that we may compare the value with the null character. If the value is not null character, you need to either increment the index or increment the pointer to move to the next character and increment the length count as well. The above is carried out in a loop.

3. Implement `mystrcat` function. Here is the idea at a bit higher level as you have already gained some experience: in a loop find the place where the null character is located in the first parameter `dest` (you may either use an index or a pointer to keep track of it). Once you find the location, you may copy the first character pointed to by the second argument `src` to the location, and repeat the copying process by increment both locations. The loop is terminated by seeing a null character, which must also be “copied”, in the second parameter (this is basically what `strcpy` needs to do)

5 Test and evaluation

1. Test the main program.

2. Add the `#include mystrlen.h` to the main program, where `mystrlen.h` is provided for the lab. Compile the main program and your implementation of `mystrlen` and test the main program. At this point, our main program will use our `mystrlen` instead of `strlen`, even though we do not modify the code of the main program (do you know why?).

3. Add the `#include mystrcat.h` to the main program resulted from the above step, where `mystrcat.h` is provided for the lab. Compile the main program and your implementation of `mystrcat` and test the main program.
At this point, our main program uses both mystrlen and mystrcat for strlen and strcat respectively (again do you know why?). If the main program behaves the same as before, you have successfully implemented strlen and strcat. Congratulations!

6 Report and documentation

A short report about things observed and things learned and understood. Properly document and indent the source code.

7 Lab submission

Get instructions from the Lab instructor.