Lab 05 - String Parsing

Out: October 6, 2014

1 Introduction

Baby Dylan and Baby Evan enter a competition to see who can parse strings better. They each have to write a tokenizer, a function to remove spaces, and a substring subroutine. But they're babies. They can't write code. It's up to you to help them out.

2 Assignment

This lab will introduce you to some simple string parsing functions. In order to help prepare you for the next lab (and for Newborn Shell, if you choose to re-use this code there), you will be implementing a few functions that manipulate strings C-style! Again, some or most of the code from this lab can be reused for the Shell assignments. Note that some of the functions you are writing are similar to library functions that already exist, but the goal of the assignment is for you to practice working with strings in C.

In this lab, you will implement the following functions:

- `char * trim_spaces( char * src );`
  This function should return the original contents of the string `src`, but without any leading or trailing whitespace. Note that any internal extra whitespace should be left alone.
- `char * next_token( char * s, char * delim );`
  Find the first sequence of characters in the delimiter set `delim` in the string `s` and return a pointer to the start of the next part of the string. Note that the next part of the string cannot include the delimiter itself. (**Hint**: this function is a helper to `tokenize()` above. To deal with the fact that you are splitting on a *set* of possible characters, you can use library functions that do most of the work for you! Take a look at the functions `strspn()` and `strcspn()` in section 3 of this document.)
- `size_t tokenize( char * buf, char * delim, char ** argv );`
  Given a string `buf`, and a set of delimiter char's to split on `delim`, this function should split `buf` on any sequence of the characters in the set `delim` and place each token into the string array, `argv`. Finally, this function should return the number of tokens read into `argv`.
- `char * find_string( char * str, char * sub );`
  We call a string a *substring* of another string if it appears exactly within the other string. This function finds and returns a pointer to the first instance of the string `sub` within `str`. If `sub` is not a substring of `str`, you should return NULL.

Note: Remember that in C, you can think of the type `char *` as any of three things, a pointer to a character, a string, or an array of characters. These are all the same!
This mean you can index into a string like an array.

2.1 Install the Stencil

To get started, run the following command in a terminal:

```
cs033_install lab05
```

This will install the stencil code (and a copy of this handout) in your *course/cs033/lab05* directory.

3 Allowed Library Functions and Resources

Read the man pages for more information about each of these functions. These are the **only external library function calls you are allowed to make** throughout this lab. There are other functions that you might be tempted to use (i.e. `strtok`), but they would make this lab trivial, and you will not be checked off if you do.

- `int isalnum(int c);`
  Checks whether `c` is an alphanumeric character.
- `int isprint(int c);`
  Checks whether `c` is a printable character (ascii characters 0 through 31 and 127 are not printable).
- `int isspace(int c);`
  Checks whether `c` is a whitespace character.
- `int is*(int c);`
  You are allowed to use any `is*` function in the `ctype.h` library. If you look up the man page for any of the above three functions, you will see a list of all of the `is*` functions.
- `void * memcpy(void * dest, void * src, size_t n);`
  Copies `n` bytes from memory area `src` to `dest`.
- `void * memset(void * dest, int c, size_t n);`
  Fills the first `n` bytes of `dest` with constant byte `c`.
- `size_t strlen(char * str);`
  Computes the length of the string `str`
- `size_t strcspn(char * str, char * reject);`
  Calculates the length of the initial segment of `str` which consists entirely of bytes not in `reject`
- `size_t strspn(char * str, char * accept);`
  Calculates the length of the initial segment of `str` which consists entirely of bytes in `accept`
- `char * strcat(char * dest, char * src);`
  Appends `src` to dest, overwriting the terminating null byte of `dest`, then adds a terminating null byte.

- `char * strchr (char * str, int c);`
  Returns a pointer to the first occurence of the character `c` in `s`
- `char * strcpy(char * dest, char * src);`
  Copies the string at `src` to the buffer at `dest`.
- `int strcmp(char * str1, char * str2);`
  Compares the two strings `str1` and `str2`. *Returns 0 if they are equal.*
- `char * strpbrk(char * str, char * accept);`
  Returns the first occurence in `str` of any bytes in the string `accept`.

4 Testing

We have included a number of tests in the main function of the stringlib.c file. These tests will test the expected functionality of this string tokenization library. While these tests do cover our basic functionality, we encourage you to write more of your own tests so that you can be sure that your tokenizer works correctly before using it in subsequent labs and projects.
Please do not modify the tests that are already in the file.

5 Getting Checked Off

Once you've finished both parts of the lab, call a TA over and have them inspect your work and run the tests. If you do not complete the lab during your lab section, you may get credit for finishing it on your own time and bringing it to TA hours to be checked off before next week's lab. Remember to read the course missive for information about course requirements and policies regarding labs and assignments.

Note that it is *highly* recommended that you finish this lab before the next one, as you will need the string parsing code for next week.