CSC161 2011S *Imperative Problem Solving*

Assignment 7: A String Library

Assigned: Friday, 8 April 2011
Due: 11:00 p.m., Wednesday, 20 April 2011

This assignment is long enough that it counts as two assignments.

Summary: In this assignment, you will build a library of useful (and not-so-useful) functions for strings. You will also explore mechanisms for testing those fucntions. sorting algorithm and explore mechanisms for testing that algorithm.

Purposes: To give you experience implementing real algorithms. To give you experience with testing. To give you more experience with simple libraries (or at least separate compilation). To give you more experience writing your own Makefiles.

Expected Time: Three to four hours.

Collaboration: I encourage you to work in groups of two or three students. However, you may work on your own or in groups of up to size four. You may discuss the assignment with anyone you wish, provided you clearly document such discussions.

Submitting: Submit tarballs of your solutions on Pioneerweb. Further details at the end of the assignment.

Warning: So that this assignment is a learning experience for everyone, I may spend class time publicly critiquing your work.

Background

As we've recently noted in class, many programs require similar components. Hence, it behooves you as programmer to build reusable *libraries* that provide those components. Since we've been working on strings, it seems particularly useful to build arrays of libraries for strings. For example, given a string of the form `"stuff:stuff:stuff:stuff"`, it can be useful to extract one part of the string.

Another component that many programs include is *sorting*: rearranging an array of values so that the array is in order. There are a number of strategies for sorting arrays and the choice of algorithm often depends on the particulars of the data.

Selection Sort

One useful and relatively simple sorting algorithm is *selection sort*. In selection sort, one repeatedly finds the index of the largest remaining element in the array and swaps it to the "end" of the subarray.

```
void
string_selection_sort (char *values[], int len)
{
  int i;
  for (i = len-1; i > 1; i--)
    {
      string_swap (values, i, string_index_of_largest (values, i+1));
    }
} // string_selection_sort
```

We assume that `string_index_of_largest (values,limit)` finds the largst value (alphabetically last) in the first `limit` elements of `values`. There are two approaches to writing `index_of_largest` : You can write it directly, or you can decompose it into two subproblems: Finding the largest element in an array and finding the index of that value.

```
int
string_index_of_largest (char *values[], int limit)
{
  return string_index_of (values, limit, string_largest (values, limit));
} // index_of_largest
```

Assignment

Here is a header file for a simple library of string utilities that we might call `string-utils.h`.

```
#ifndef __STRING_UTILS_H__
#define __STRING_UTILS_H__

/**
 * string-utils.h
 *   Declarations for a variety of useful and fun string utilities.
 */


// +------------------+-----------------------------------------------
// | Fun with Strings |
// +------------------+

/**
 * Procedure:
 *   the_name_game
 * Parameters:
 *   name, a string
 * Purpose:
 *   Print the lyrics to one verse of the name game.
 * Produces:
 *   [Nothing; called for the side effect.]
 * Preconditions:
 *   name is not empty.
 * Postconditions:
 *   A verse of the name game has been printed to standard output.
 * Practica:
 *   the_name_game ("Nick");
 *     Nick!
 *     Nick, Nick bo Bick Bonana fanna fo Fick
```

```
 *     Fee fy mo Mick, Nick
 *   the_name_game ("Shirley");
 *     Shirley!
 *     Shirley, Shirley bo Bhirley Bonana fanna fo Fhirley
 *     Fee fy mo Mhirley, Shirley!
 *   the_name_game ("Akshay");
 *     Akshay!
 *     Akshay, Akshay bo Bakshay Bonaa fanna fo Fakshay
 *     Fee fy mo Makshay, Akshay!
 * Props:
 *   Shirley Ellis performed the best-known version of the name game.
 *   She also co-wrote the song.
 * Problems:
 *   (1) But if the first two letters are ever the same,
 *       You drop them both and say the name like
 *       Bob, Bob drop the B's Bo ob
 *       For Fred, Fred drop the F's Fo red
 *       For Mary, Mary drop the M's Mo ary
 *       That's the only rule that is contrary.
 *   (2) Ms. Ellis is inconsistent on the dropping of letters.  For
 *       example, she drops only the F in Fred, but both the S and h
 *       in Shirley.  We drop only the first letter
 *
 */
void the_name_game (char *name);

/**
 * Procedure:
 *   print_sequential_prefixes
 * Parameters:
 *   str, a string
 *   ch, a character
 * Purpose:
 *   Print the substrings of str that start with ch.
 * Produces::
 *   [Nothing; called for the side effect]
 * Preconditions:
 *   [No additional]
 * Postconditions:
 *   The output now contains the substrings.
 * Practica:
 *   print_sequential_prefixes ("twas brillig and the slithy toves did gyre", 't');
 *      twas brilling and the slithy toves did gyre
 *      the slighthy toves did gyre
 *      thy toves did gyre
 *      toves did gyre
 */
void print_sequential_prefixes (char *str, char ch);


// +-------------------------+---------------------------------------
// | Basic String Utilities  |
// +-------------------------+

/**
 * Procedure:
 *   string_reverse
```

```
 * Parameters:
 *   str, a string
 * Purpose:
 *   Reverse the string in place.
 * Produces:
 *   rev, the reversed string
 * Preconditions:
 *   [No additional]
 * Postconditions:
 *   Let len be the length of str and STR be its original value.
 *     rev[i] = STR[len-1-i] for all i from 0 to len-1.
 */
char *string_reverse (char *str);

/**
 * Procedure:
 *   column
 * Parameters:
 *   str, a string
 *   sep, a character
 *   col, an integer
 *   result, a string
 * Purpose:
 *   Grab the col'th column from str, given that columns are separated
 *   by sep.  Copy that column into result.
 * Produces:
 *   ok, an integer
 * Preconditions:
 *   result is big enough to hold the substring.
 *   col >= 0
 * Postconditions:
 *   If str cotains at least col columns, result contains the value in
 *     column col and ok is 1.
 *   Otherwise, result is unmodified and ok is 0.
 * Practica:
 *   column ("alpha:beta:gamma", ':', 0, foo);  // foo now contains "alpha"
 *   column ("alpha:beta:gamma", ':', 1, foo);  // foo contains "beta"
 *   column ("alpha:beta:gamma", ':', 2, foo);  // foo now contains "gamma"
 *   column ("alpha:beta:gamma", ':', 3, foo);  // returns 0
 *   column ("a:b:c", ',', 0, foo);             // foo now contains "a:b:c"
 *   column ("aardvark,banana", ',', 0, foo);   // foo now contains "aardvark"
 *   column ("aardvark,banana", ',', 1, foo);   // foo now contains "banana"
 */
int column (char *str, char sep, int col, char *result);


// +-------------------------+----------------------------------------
// | String Array Utilities  |
// +-------------------------+

/**
 * Procedure:
 *   string_swap
 * Parameters:
 *   strings, an array of strings
 *   i, an integer
 *   j, an integer
```

```
 * Purpose:
 *   Swap the strings in positions i and j.
 * Produces:
 *   [Nothing; called for the side effect]
 * Preconditions:
 *   0 <= i,j < length of strings
 * Postconditions:
 *   If STRINGS has the old value of strings, then
 *     strings[i] = STRINGS[j]
 *     strings[j] = STRINGS[i]
 *     strings[k] = STRINGS[k] for all k != i,j
 */
void string_swap (char *strings[], int i, int j);

/**
 * Procedure:
 *   string_largest
 * Parameters:
 *   strings, an array of strings
 *   limit, an integer
 * Purpose:
 *   Find the largst string in the first limit strings in strings.
 * Produces:
 *   largest, a string
 * Preconditions:
 *   strings has length at least limit
 *   limit >= 1.
 * Postconditions:
 *   strcmp (largest, strings[i]) >= 0 for all 0 <= i < limit
 *   strings is unmodified
 *   There exists a j s.t. largest == strings[j].
 */
char *string_largest (char *strings[], int limit);

/**
 * Procedure:
 *   string_index_of
 * Parameters:
 *   strings, an array of strings
 *   size, an integer
 *   str, a string
 * Purpose:
 *   Find the index of str in the first size elements of strings.
 * Produces:
 *   index, an integer
 * Preconditions:
 *   strings has length at least size.
 * Postconditions:
 *   If exists i s.t., strcmp (strings[i], str) == 0, then index is one such i.
 *   Otherwise, i is -1.
 */
int string_index_of (char *strings[], int size, char *str);

/**
 * Procedure:
 *   string_sort
 * Parameters:
```

```
 *   strings, an array of strings
 *   size, an integer
 * Purpose:
 *   Sort the first size elements of strings.
 * Produces:
 *   [Nothing; Called for the side effect]
 * Preconditions:
 *   size <= length of strings
 * Postconditions:
 *   The final version of strings is a permutation of the original
 *     version of strings.
 *   strcmp (strings[i], strings[i+1]) <= 0 for all i < size-1.
 */
void string_sort (char *strings[], int size);


#endif // _STRING_UTILS_H
```

Create the following files.

- `the-name-game.c` - defines the `the_name_game` function.
- `name-game.c` - provides a `main` that takes a series of names from the command line and prints out the verses for those names.out the verses for those names.
- `shirley.c` - provides a `main` that prints out the verses for Shirley, Lincoln, Bob, Marsha, and Raynard.
- `print-sequential-prefixes.c` - defines the `print_sequential_prefixes` function.
- `psp-examples.c` - provides a `main` that shows the results of some calls to `print_sequential_prefixes`.
- `string-reverse-utests.c` - provides a `main` that does unit tests for `string-reverse`.
- `string-reverse.c` - defines the `string_reverse` function.
- `reverser.c` - provides a `main` that reads text from standard input and prints out the reverse of that text. You may assume that no line has more than 128 characters. You may reverse individual lines or the complete text (the latter is more interesting, but slightly harder).
- `column-tests.c` - provides a `main` that does unit tests for `column`.
- `column.c` - defines the `column` functions.
- `column-ui.c` - provides a `main` that takes an integer on the command line and selects that column from standard input. You can assume that the lines of standard input are separated by colons and that no line is more than 128 characters.
- `swap-utests.c` - provides a `main` that does unit tests for `string_swap`.
- `swap.c` - defines `string_swap`.
- `largest-utests.c` - provides a `main` that does unit tests for `string_largest`.
- `largest.c` - defines `string_largest`.
- `index-of-utests.c` - provides a `main` that does unit tests for `string_index_of`.
- `index-of.c` - defines `string_index_of`.
- `sort-utests.c` - provides a `main` that does unit tests for `string_sort`.
- `sort.c` - provides a `main` that sorts the strings provided on the command line.
- `selection-sort.c` - an implementation of `string_sort` using the selection-sort algorithm.
- `merge-sort.c` - an implementation of `string-sort` using the merge-sort algorithm.

Create a `Makefile` that ties it all together. I should be able to create the following targets:

- `all` - builds all the applications (not the unit tests), including `name-game`, `shirley`, `reverser`, `column`, `selection-sort`, and `merge-sort`. (The latter two should link `sort.o` with the appropriate one of `selection-sort.o` or `merge-sort.o` as well as any other necessary files.)
- `test` - builds and runs all the unit tests. Make sure to have separate unit tests for `selection-sort` and `merge-sort`>
- `clean` - cleans out all the `.o` and Emacs backup files (and any other backup files your editor might create).
- `distclean` - cleans out all the .o, backup files, applications, and unit tests.
- `package` - builds a tarball of all the useful files.

Hints

In your unit tests for `selection_sort`, you will find it useful to compare arrays. Here is a procedure for comparing arrays.

```
/**
 * Procedure:
 *   string_array_equal
 * Parameters:
 *   a1, an array of strings
 *   a2, an array of strings
 *   len, an integer
 * Purpose:
 *   Determine if the first len elements of a1 and a2 are equal
 * Produces:
 *   is_equal, an integer (representing a Boolean)
 * Preconditions:
 *   len >= 0.
 *   a1 contains at least len elements.
 *   a2 contains at least len elements.
 * Postconditions:
 *   If there exists an i, 0 <= i < len s.t. a1[i] != a2[i],
 *     returns 0.
 *   Otherwise, returns 1.
 */
int
string_array_equal (char *a1[], char *a2[], int len)
{
  int i;
  for (i = 0; i < len; i++)
    {
      if (strcmp (a1[i], a2[i]) != 0)
        return 0;
    } // for
  // No unequal elements.  They match!
  return 1;
} // string_array_equal
```

Submitting Your Homework

Using `script`, build a log of sample runs of your programs. Your script should be stored in the file `typescript`.

Create a `README` to insert any information you'd like to provide to me or the grader.

Using `make package`, build a tarball of all the important files.

Test your package by untaring it somewhere else and running `make all` and `make test`.

Submit the tarball on Pioneerweb under the Assignment 7 link.
