Assignment 5: Sorting Out Sorting

Assigned: Friday, 8 October 2010
Due: 11:00 p.m., Wednesday, 13 October 2010

Summary: In this assignment, you will implement a famous 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 experience with simple libraries (or at least separate compilation).

Expected Time: Two to three 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: Email me your answers. See further details below.

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.

One 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.

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.

```c
void selection_sort (int values[], int len)
{
    int i;
    for (i = len-1; i > 1; i--)
    {
        swap (values, i, index_of_largest (values, i+1));
    }
} // selection_sort
```

We assume that `index_of_largest (values,limit)` finds the largest value 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.

```c
int
index_of_largest (int values[], int limit)
{
    return index_of (values, limit, largest (values, limit));
} // index_of_largest
```

**Assignment**

a. Create `array_utils.h`, which contains declarations for `swap`, `largest`, `index_of`, `index_of_largest`, `selection_sort`, and any other procedures you expect to write. (Don’t worry if you can’t come up with them all. You can always change the header later.)

b. Write unit tests for `swap`, `largest`, and `index_of`. You should have three files: `utest_swap.c`, `utest_largest.c`, and `utest_index_of.c`.

c. Write `swap`, `largest`, and `index_of`. All of these should be in a single source file, `array_utils.c`. Use your unit tests to ensure that your code is correct.

d. Write unit tests for `index_of_largest`. Use these unit tests to make sure that the provided implementation of `index_of_largest` is correct. If it is not, correct it.

e. Write an interactive tester for `selection_sort`. Your tester should read up to 1024 integers from standard input and print them out in sorted order. Call your tester `sort.c` (executable `sort`). Use the interactive tester on some inputs you find useful. Correct any errors you detect in `selection_sort`.

f. Write a systematic unit tester for `selection_sort`, `utest_selection_sort.c`. Using this tester, identify and correct any remaining errors in `selection_sort`.

**Hints**

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

```c
/**
 * Procedure:
 *   array_equal
 * Parameters:
 *   a1, an array of integers
 *   a2, an array of integers
 *   len, an integer
 * Purpose:
 *   Determine if 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 \( i, 0 \leq i < \text{len} \) s.t. \( a1[i] \neq a2[i] \),
* returns 0.
* Otherwise, returns 1.
*/

```c
int array_equal (int a1[], int a2[], int len)
{
    int i;
    for (i = 0; i < len; i++)
    {
        if (a1[i] != a2[i])
            return 0;
    } // for
    // No unequal elements. They match!
    return 1;
} // array_equal
```

**Submitting Your Homework**

Using script, build logs of sample runs of your programs.

Put those logs, your source code, and any other files you deem appropriate in a directory called `username.hw5`.

Make a tarball of that directory.

Submit that tarball as an attachment to an email message entitled should be titled CSC161 Assignment 5.