CSC161 2010F *Imperative Problem Solving*

Assignment 7: Sorting Out Sorting, Again

Assigned: Friday, 5 November 2010
Due: 11:00 p.m., Wednesday, 10 November 2010
Revised due date: 7:00 p.m. Friday, 12 November 2010

Summary: In this assignment, you will experiment with sorting and with ways of writing somewhat generic code in C.

Purposes: To ground your learning of some sorting algorithms. To give you experience with a useful C technique.

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: Email me a tarball of your important files (your `.c` files, your `.h` files, your `Makefile`, and anything else you deem appropriate).

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

Writing Generic Code in C

As we saw in a recent class, because C is typed, it is difficult to write "generic" code in C. However, it is not impossible. In particular, if we create a `.c` file that relies on some definitions for types and names, then we can set those types and names differently, to obtain different versions of the same program.

For example, consider the problem of swapping two values in an array. We've written a procedure to swap two elements in an array of integers as follows.

```
void
int_aswap (int a[], int i, int j)
{
  int temp = a[i];
  a[i] = a[j];
  a[j] = temp;
} // int_aswap
```

Similarly, if we wanted to write something to swap two values in an an array of doubles, we might write

```
void
double_aswap (double a[], int i, int j)
{
  double temp = a[i];
  a[i] = a[j];
  a[j] = temp;
} // double_aswap
```

Note that these procedures differ only in (1) the prefix we apply to `aswap`, (2) the declaration of `a`, and the declaration of `temp`. We might therefore write a generic version of this as follows, with the understanding that a programmer can substitute in the type and prefix.

```
/** swap-generic.c *//
void
PREFIX(aswap) (TYPE a[], int i, int j)
{
  TYPE temp = a[i];
  a[i] = a[j];
  a[j] = temp;
} // generic awap
```

Why did we use `PREFIX` rather than `TYPE`? Because in some cases, the type may not match the prefix we want to use. For example, for strings, we might want the prefix to be `string_`, but the type will be `char *`.

Now, here's a cool thing. We can use the C preprocessor to take the place of the programmer. In particular, we can define `PREFIX` to put the `float_` (or whatever) on the front, and `TYPE` to be `float` (or whatever). So, in order to get `float_aswap`, we just write

```
/** float-swap.c */
#define PREFIX(FUN) float_ ## FUN
#define TYPE float
#include "generic-swap.c"
```

The C pre-processor will then automatically turn this into

```
void
float_aswap (float a[], int i, int j)
{
  float temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}
```

You can try this yourself. The files are in the example directory.

Sorting

As we've discussed, there are a wide variety of sorting algorithms. Some of the more efficient sorting algorithms rely on a *divide and conquer* strategy. For example, in merge sort, we divide the array into two halves, sort the two halves, and then merge them together into a single sorted array. Similarly, in Quicksort, we rearrange the elements into small and large elements using a random pivot, sort the two

halves, and we're done.

Assignment

Your job is to build a variety of files that use the generic technique to implement three sorting routines (merge sort, Quicksort, and one sorting routine of your choice) for three types (integers, doubles, and strings).

In particular, you should create the following files:

- `generic-sort.h`, a generic declaration of a sort routine; this header should declare `PREFIX(sort)(TYPE vals[], int size)`. Programs will not include `generic-sort.h` directly. Rather, the type-specific header files will declare it.
- `int-sort.h`, which declares the macros and constants for sorting arrays of integers and then includes `generic-sort.h`. Your file will look something like the following:

```
#ifndef __INT_SORT_H__
#define __INT_SORT_H__

/**
 * int-sort.h
 *   Declarations of sorting routines for arrays of integers.
 */
#define PREFIX(FUN) int_ ## FUN
#define TYPE int
#define MAY_PRECEDE(X,Y) (X <= Y)
#include "generic-sort.h"
#endif // __INT_SORT_H__
```

- `double-sort.h`, a similar file for arrays of doubles.
- `string-sort.h`, a similar file for arrays of strings.
- `generic-merge-sort.c`, a generic implementation of merge sort. Your file will begin something like the following.

```
/**
 * generic-merge-sort.c
 *   A generic implementation of merge sort.  To use this implementation,
 *   define PREFIX(FUN), TYPE, and MAY_PRECEDE.
 */
void
PREFIX(sort) (TYPE vals[])
{
  ...
}
```

- `generic-Quicksort.c`, a generic implementation of Quicksort.
- `generic-`*`other`*`-sort.c`, a generic implementation of some other sorting routine (insertion sort, selection sort, Shell sort, heap sort, bubble sort, permutation sort, etc.)
- `int-merge-sort.c`, a specification of `generic-merge-sort.c` to arrays of integers.
- `double-merge-sort.c`, a specification of `generic-merge-sort.c` to arrays of integers.
- `string-merge-sort.c`, a specification of `generic-merge-sort.c` to arrays of strings.

- `int-Quicksort.c`, a specification of `generic-Quicksort.c` to arrays of integers.
- `double-Quicksort.c`, a specification of `generic-Quicksort.c` to arrays of integers.
- `string-Quicksort.c`, a specification of `generic-Quicksort.c` to arrays of strings.
- `int-`*`other`*`-sort.c`, a specification of `generic-`*`other`*`-sort.c` to arrays of integers.
- `double-`*`other`*`-sort.c`, a specification of `generic-`*`other`*`-sort.c` to arrays of integers.
- `string-`*`other`*`-sort.c`, a specification of `generic-`*`other`*`-sort.c` to arrays of strings.
- Any files that you consider appropriate for testing your procedures. You are likely to want to build some unit tests and some interactive tests.
- A Makefile that ties everything together.
