CMSC 202	**Homework 4**	Spring 1996
Sections 101, 102 & 103	**due: Thurday, 28 March**	Handout #15

This assignment is designed to give you practice using lists and functional parameters.

You will write two programs to read data from a file, then print those data in sorted order. The sorting will be done in two different ways in the two programs. The first program will insert the data one item at a time into an ordered list. The second program will insert all of the data into an array, then use the library routine `qsort` on the array to order the items. `Qsort` is described on page 329 of the text. There will be lots of common code between the two programs if you plan well (in fact, the two programs may share source files).

Each data point in the input file has three components: an `int`; a `float`; and a `string`. Each data point is to be stored in a `struct` of three fields. You will therefore be creating a list of pointers to `struct`s in one program, and an array of pointers to `struct`s in the other.

The Data File

The data file is made up of comment lines (which begin with a `#` character), blank lines, and data lines. The first data line contains an integer indicating the number of data entries in the file. The remaining data lines contain the entries, which comprise an integer, a real, and a string, all on the same line, separated by commas. A sample data file is available as `~mayfield/hw4.dat`. Here is a listing of its contents:

```
# hw4.dat
# Data for use with HW-4, CMSC-202, Spring 1996

# The first non-comment line contains the number of entries
# in the rest of the file.
# Each of these entries is a comma-separated list of three
# elements -- an int, a float, and a string.  White space on
# a line is to be ignored.

# Blank lines are allowed in the file.

4
27, 3.14, Tom
19, 4.56, Jeff
# Comments may appear anywhere.

3,  9.12, Al
30, 1.13, Mary
```

You can create other sample data files to test your program by running the command `~mayfield/make-h4-datafile` and placing the output in a file.

The Programs

The outward appearance of the two programs will be identical. The internal difference is that one inserts into a sorted list, the other sorts an array. Here is how the programs look when run on the data file **`hw4.dat`**. Each time the program is called, a different comparator function is chosen (but not all the comparator possibilities are explored in this script). User input is shown in *italics*:

```
umbc10[101] hw4a
Enter the name of the data file: hw4.dat
Enter the number of the desired ordering:
Select 1 to sort integers in ascending order.
Select 2 to sort reals in ascending order.
Select 3 to sort strings in alphabetic order.
Select 4 to sort integers in descending order.
Select 5 to sort reals in descending order.
Select 6 to sort strings in reverse alphabetic order.
Your choice: 1
(3 9.120000 Al)
(19 4.560000 Jeff)
(27 3.140000 Tom)
(30 1.130000 Mary)

umbc10[102] hw4a
Enter the name of the data file: hw4.dat
Enter the number of the desired ordering:
Select 1 to sort integers in ascending order.
Select 2 to sort reals in ascending order.
Select 3 to sort strings in alphabetic order.
Select 4 to sort integers in descending order.
Select 5 to sort reals in descending order.
Select 6 to sort strings in reverse alphabetic order.
Your choice: 3
(3 9.120000 Al)
(19 4.560000 Jeff)
(30 1.130000 Mary)
(27 3.140000 Tom)

umbc10[103] hw4a
Enter the name of the data file: hw4.dat
Enter the number of the desired ordering:
Select 1 to sort integers in ascending order.
Select 2 to sort reals in ascending order.
Select 3 to sort strings in alphabetic order.
Select 4 to sort integers in descending order.
Select 5 to sort reals in descending order.
Select 6 to sort strings in reverse alphabetic order.
Your choice: 5
(3 9.120000 Al)
(19 4.560000 Jeff)
(27 3.140000 Tom)
(30 1.130000 Mary)
```

```
umbc10[104] hw4a
Enter the name of the data file: hw4.dat
Enter the number of the desired ordering:
Select 1 to sort integers in ascending order.
Select 2 to sort reals in ascending order.
Select 3 to sort strings in alphabetic order.
Select 4 to sort integers in descending order.
Select 5 to sort reals in descending order.
Select 6 to sort strings in reverse alphabetic order.
Your choice: 6
(27 3.140000 Tom)
(30 1.130000 Mary)
(19 4.560000 Jeff)
(3 9.120000 Al)
```

Your program should print an error message and exit:

- if the data file cannot be opened.
- it the user enters an illegal menu number.

You may assume that the data file is well–formed, so you need not check for errors in it.

Problem 1

The program **hw4a** performs the following tasks:

- prompts for and opens a data file;
- prompts for a sorting order;
- reads the entries in the data file one at a time into an ordered list; and
- prints the list data.

Problem 2

The program **hw4b** performs the following tasks:

- prompts for and opens a data file;
- prompts for a sorting order;
- reads the number of entries from the data file;

- dynamically allocates an array of the correct size;
- reads the entries in the data file into the array (unsorted);
- runs the `qsort` library routine on the array to achieve the desired order; and
- prints the array data.

Using the Lists Package

To use the course lists package, you will need to:

- `Typedef` a new type called `list_item_type`, which will be the type of the objects in your list. **WARNING:** `list_item_type` may not be larger than a pointer. That means that a `struct` with three fields is not a legal `list_item_type`; you should use a pointer to such a `struct`. To make sure that the size of your `list_item_type` is acceptable, you can put a call to the function `check_list_item_size()` at the top of your main program.
- Include the file `"lists.h"`. Your inclusion of this file must come *after* your definition of `list_item_type`.
- Compile your program so that it uses the 202 libraries. To do so, you will need add several items to your Makefile. Lines of the Makefile that create .o files will need to include '`-I ~mayfield/lib202`' immediately after the `cc` command, *e.g.*:

  ```
  cc -I ~mayfield/lib202 -g -c myfile.c
  ```

  Lines of the Makefile that build executables will need to include '`-L ~mayfield/lib202 -l202`' immediately after the list of .o files, *e.g.*:

  ```
  cc -o myprog myfile.o myfile_aux.o -L ~mayfield/lib202 -l202
  ```

Submission

To submit your files, use the `submit` program. For example, to submit the file `hw4a.c`, enter:

```
submit cs2021 hw4 hw4a.c
```

You must submit all files necessary to successfully make both parts of your homework. You must submit a Makefile. The Makefile must cause creation of the executable `hw4a` when '`make hw4a`' is entered. It must cause creation of the executable `hw4b` when '`make hw4b`' is entered. It must cause creation of both executables when either '`make all`' or just '`make`' is entered. Your executables must be named `hw4a` and `hw4b`. Aside from these restrictions, you may name your files as you wish.