

Assignment 12: Sorting Files

Complete assignment due: Thursday, Nov 20th, 9:00 p.m.

This assignment has nothing whatsoever to do with matching words :-).

Write a C program that takes the lines in one or more text files and sorts them. For each file, print the name of the file. Then print each line of the file in sorted order. Name the program **sortFile**.

```
sortFile [-bir] file [file ...]
```

There will be three optional arguments:

-i, --insensitive: When present, the sort will be case insensitive.

-r, --reverse: When present, the sort will be in reverse alphabetic order.

-b, --both: When present, print two sorts. The first will be the lines of the file in alphabetic order. Then print a blank line, a row of 70 asterisk's, and another blank line. Then print the file in reverse alphabetic order.

After printing the results for each file, print a blank line.

Exit with a status of **0** after printing all the files.

Linked List:

Use a linked list of **struct**'s in your solution. There is no upper-bound on the number of lines that may be present in each file; thus, a linked list will allow your program to work with arbitrarily large files.

For the lines in each file, each line will not be longer than **160** characters, including the newline character at the end. In the **struct** that you create to hold each line, do not put an array of characters of size **160**. Instead, put a pointer in the **struct**. For each line you read from the file, dynamically allocate space of exactly the right size to hold the line. There are two reasons for this requirement: First, most of the lines in a file will be contain much less than **160** characters, so allocating the exact right amount saves memory space. Second, the idea is to give you practice at managing dynamically allocated memory.

Memory Leaks:

Your program will free all memory that is dynamically created during execution. The **valgrind** program on lectcura can be used for this purpose. It has various options. Without options, **valgrind** reports memory leaks. Note that the output from **valgrind** goes to **stderr**, not to **stdout**.

A basic use for this assignment:

```
valgrind sortFile -i file1.txt file2.txt
```

If **valgrind** reports there are memory leaks, you may find the **--leak-check=full** option to be useful.

Grading Notes:

Your program will be graded in part in the same way we have graded previous programs. In addition, your program will need to free all dynamically allocated memory. Your program must use dynamic linked lists. Each element of the list will hold one line of the file. The line will be in a dynamically allocated array.

We will use code inspection and **valgrind** to examine how you create, manage, and free the linked lists in your program.

Make and Functions:

As with the previous two assignments, you need to divide your code into functions. Place the functions into at least two C files and at least one header file.

Provide a *Makefile* that supports incremental compilation of your code. The *Makefile* will have a **sortFile** target. We will type **make sortFile** to create your **sortFile** executable. The *Makefile* will also support the target: **make clean**. This will remove all **.o** files created by during compilation of your program.

Turnin: Use **turnin** to turn in all the files for your program. This includes the **Makefile**, all the C program files, and the header file(s). The command is:

```
turnin 352assign12 Makefile ...
```

Special note: If you are using your **Makefile** from assignment 11, be sure to change the assignment name in the **Makefile** **turnin** to be **352assign12**.

See the man page for the **turnin** program for details on what **turnin** can do and how you can confirm that your file was turned in.