csci-e28
Project 0: more03
Due: This Sunday Night at 11:59PM
Last Update: Sun 2 Feb 2025 15:40 PM

Purpose of Assignment

There are three main purposes of this assignment.

The first is to help you see if you are prepared for the course before the deadline to drop for a refund. If you find this project very difficult, you will find the course very demanding. Which does not mean you should not take the course. We can help you figure out, based on your problems, what you need to learn and whether that is realistic.

The second purpose is to show you the typical model of a solution. This program has two files; you have to be able to work with multi-file programs. This project has a Makefile; you have to submit a Makefile with each assignment. This project has a design document, called Plan. We require a plan for each assignment.

The final purpose is to make sure you are able to use the system to compile and run C programs.

Introduction

In the first class, we wrote two versions of the Unix *more* file-viewing tool. The first version, more01.c, read user commands from standard input and displayed files listed on the command line. It had a serious problem: it did not work correctly when used at the end of a pipeline, as in:

```
last | more01
```

In this pipeline, more01, seeing no files listed on the command line, reads data from standard input. The problem is that the program reads user commands from standard input, too.

The solution, coded in more02.c, is to have the program read user commands from the terminal device regardless of source of text to display. Every Unix system has a file called */dev/tty* which is an interface to the user's keyboard and display. Any data read from this file comes from the user's keyboard. Any data written to this file appears on the user's display.

more02.c solves this problem well.

Problems with more02.c

But more02.c has other problems. Specifically:

[a] The program is hard-coded for a 24-line display, once a standard. These days, a terminal window can be resized to any number of rows. A better version will ask the system for the number of lines in the terminal window and use that number to display a `page´ of text.

[b] The program does not adapt to window sizes while it is running. Say the terminal window starts with 24 lines, but after seeing the first page of text, the user resizes the window to 38 lines then presses the space bar. A better version will adapt to the current size of the screen, even if that size changes after the program starts.

[c] The program accepts user input SPACE, Enter, and **`q`** to mean *see next page*, *see next line*, and *quit*. But, because of the default settings of the Unix keyboard input sytem, the user has to press Enter to send the keystrokes to the program. (The reason is that the system allows the user to correct typing errors before transmitting a line of text.) Also, more02 displays the q or space char, while regular more does not. A better version of the program will, like the regular system versions of more, receive each key as it is pressed, and not display the key.

[d] The program assumes printing one input line uses one line on the output screen. But, if a line is longer than the screen width, that line uses more than one line on the output screen. A better version will handle lines longer than screen width correctly. Test this with the system version of more to see what 'correctly' means. Experiment with a file with long lines to see exactly how regular more handles these. Your program should behave just like regular more. Do not worry about tabs.

[e] more02 leaves the more? prompt on the screen when the user presses the Enter key.

It also leaves the prompt on the screen when the user presses space, but it can be scrolled off the top. A better version will not have leave more? prompts on the screen.

The Assignment

Write a version of more, called *more03.c* based on the program *more02.c* we looked at in class. Also, answer the design question posed below in The Details.

Add features [a], [b], [c], [d], [e] listed above to more02.c using some functions we provide in a separate file called *termfuncs.c*.

One of the functions tells the caller the dimensions of the terminal window. The other function reads one key from the keyboard without requiring the user to press Enter.

Note that get_term_size returns an error code if that call fails. Make sure your version checks for and does something sensible in case of error.

Compare Your Program to the System Version

Compare your more03 to the standard version of more on the system. To see what your program should do, simply run the regular version of more and see how it behaves. Try to get your version to behave the same way.

The Details

Follow these steps to get started.

1. Get an account on the Harvard server .
   Get a Harvard Key at https://key.harvard.edu .
2. Then use that account name and password to login to your account on cscie28.dce.harvard.edu . If you have Linux or OSX on your machine, open a terminal and use ssh to connect. On Windows, you can use ssh from the cmd prompt windown. On Windows you can also download puTTY or other ssh client and use that program to connect to cscie28.dce.harvard.edu.
3. Browse to cscie28.dce.harvard.edu/~dce-lib215 and select the "Projects" link. Read the documents linked at the top about procedures and policies for writing and submitting programming projects.
4. Make a directory for the assignment called more03. Type:

   ```
   mkdir more03
   cd more03
   ```

5. Copy some starter files to this directory: Type:

   ```
   cp ~dce-lib215/hw/more03/files/* .
   ```

   note: there is a dot at the end of that command
6. Now copy more02.c to a new file: more03.c
7. Examine files: termfuncs.c, termfuncs.h, Makefile, Plan, README, and more02.callgraph.
8. Edit more03.c to add the features required. At the top of more03.c, add the line:

   ```
   #include "termfuncs.h"
   ```

   Try one change at a time; do not do all at once. It is easier to debug one problem than to debug three at once.
9. After each change, build the tool again, and test it: Type:

   ```
   make
   ./more03 /etc/passwd *.callgraph
   last | ./more03
   ```

   Please make more and better test data. Submit some additional test data with your code. **Important**: Test your program carefully. The original version has two subtle bugs (two I know about). State those bugs in the README file. Correct those bugs. What are possible users errors, special cases, "edge-cases"?
10. **Design Question**: The system version of more displays the percentage of the file shown. For example, type more /etc/passwd to see the more prompt contains a percentage. You do not have to add this feature to your program. Instead, describe in your Plan file an outline of how you would add this feature. What information do you need to do this, and how does the program change to provide this feature? Again, you do not have to implement the feature, just discuss design plans.
11. Update the files called *Plan* to reflect accurately the design and logic of your solution. Include in *Plan* any problems or clever ideas you want to point out. Also include your design ideas for adding the percentage display.
12. Update the file called *README* .
13. Create a sample run. Type:

```
script
make clean
make
./more03 more03.c
last | ./more03
exit
```

Submitting Your Work

When done, submit your work: Type:

```
~dce-lib215/handin more03
```

Note: you must be in the directory with the files you are submitting.

Note on Design/Documentation

The sample starter code, more02.c, is a good example of the coding standards for commenting, layering, and the format we expect in your code. Please use this as a guide. Note: the commenting in your code must be sufficient but does not have to be as extensive as that in the sample code.