1 Objectives

The objectives of this lab are for students to get:

1. More practice with C programming in general.
2. Experience with application-level file management.

2 Requirements

2.1 Big Picture

Create a program that copies an arbitrary text file (i.e., any text file of any size) while changing any lower-case characters to upper-case characters during the copy.

2.2 Requirements

1. Create a program called `mycopy` that takes two text files as inputs:

   ```
   mycopy source destination
   ```

   Where the name of the source file is found in argv[1] (i.e., the file to be copied), and the name of the destination file is found in argv[2].
2. You should get two files on the command-line; otherwise, the program shall print a useful error message and exit with a non-zero value.
3. If the source and destination file names are the same, then the program shall print a useful error message and exit with a non-zero value. [Note that comparing the two input strings will be good enough for this exercise, but in general would not normally be good enough. For example, this simple test would fail to detect that they refer to the same file if one string is an absolute path to a file, while the other string is a relative path to the same file].
4. If the **source** file does not exist, then the program shall print a useful error message and exit with a non-zero value.
5. If the **destination** file already exists, then the program shall print a useful error message and exit with a non-zero value. **When testing, please be careful about the file name you give for the destination because if you do this wrong you may lose the content of that file.**
6. If the program cannot create the destination file, then it shall print a useful error message and exit with a non-zero value.
7. The program shall copy the contents of the source file into the destination file while also changing any lower-case character to an upper-case character before it is written.
8. The program must use the standard C file functions (i.e., fopen, fread, fwrite, fclose), **not** the Unix file functions.
9. The program must use the buffer approach to file I/O rather than reading from and writing to a file one character at a time.

10. The program shall also keep track of the following information while performing its job:
    a. The number of characters copied.
    b. The number of characters that were changed during the copy.
    c. The number of lines in the file that were copied. For our purposes, we will simplify the interpretation of this requirement to be the number of times '\n' is seen in the file.
    d. The number of punctuation characters that were copied.
11. The execution and output shall look like the following example.

```
./mycopy test1 test2
Number of characters copied  = 13
Number of characters changed = 8
Number of lines in the file  = 1
Number of punctuation chars  = 1
```

    The above is the result of copying a file named "test1" that contained the content of "Hello World!" (which invisibly contained a '\n' at the end).
12. All error messages **must** go to `stderr`. Hint: use `perror` when errno is set to something other than zero, and `fprintf` for displaying all other error messages.
13. The implementation file shall be called `mycopy.c`.
14. Other C-library functions that may be useful:
    a. `islower`
    b. `isupper`
    c. `toupper`
    d. `ispunct`

2.3 Notes

1. You can use the following command to see all the characters in a file (even the "hidden" characters):
   ```
   hexdump -c filename
   ```
   If the file is large, then pipe the output through `more`:
   ```
   hexdump -c filename | more
   ```
2. You can compare two text files by using the following command:
   ```
   diff -i file1 file2
   ```
   If the files are identical, then no output is given. (The "-i" tells diff to ignore case differences).

3 Submission

Post `mycopy.c` on Sakai by the due date.

4 Grading

Your grade will be based on the following **guidelines**:

1. Compiles without errors or warnings (20).
2. Proper memory handling, such as valgrind reporting no errors, using NULL appropriately, and no segmentation faults (10).
3. Handles the following error conditions correctly (3 points per condition = 24):
    a. ./mycopy (no files given)
    b. ./mycopy file1 (only one file given)
    c. ./mycopy file1 file1 (source and destination are the same)
    d. ./mycopy file1 file2 file3 (too many files given)
    e. ./mycopy /etc/shadow file1 (no permissions to read /etc/shadow)
    f. ./mycopy file1 /etc/file1 (no permission to create files in /etc)
    g. ./mycopy zzzzzzzzzzz file4 (nonexistent source file)
    h. (destination file already exists)
4. Performs as specified:
    a. The copied file is an exact duplicate of the original file (ignoring upper-case/lower-case differences). This includes a file with no content, as well as files for which you only have read access. (10)
    b. The displayed statistics about the copying are correct (as described in Section 2.2, requirements 10 and 11). (5)
    c. All error messages go to stderr. All other output goes to stdout. (5)
    d. In general, the program performs as described in Section 2.
5. A code review shows the following:
    a. You close open files before exiting the program (even if the program terminates due to an error). (6)
    b. Only standard C file functions are used to read and write the files (rather than using Unix calls to read and write files). (5)
    c. A buffer is used to read and write files (rather than reading and writing one character at a time). (5)
    d. No deviations from the Style Guide. (10)
6. I reserve the right to deduct points for other reasons.