Introduction

The purpose of this assignment is to review basic Unix and C skills and help you figure out if you are prepared for this course. Work through the following exercises. You should of course feel free to refer to any Unix/C books and/or on-line documentation you like. You should find the C exercises pretty easy. If you don't, think carefully before enrolling. You should find the Unix exercises easy or you should be able to locate the information in the manuals. If you don't, find a good Unix book and start exploring.

Solutions to these problems will be available at the second lecture.

The Exercises

1. Explain the purpose of the following Unix commands: `ls, cat, rm, cp, mv, mkdir, cc`.

2. Using your favorite editor, create a small text file. Use `cat` to create another file consisting of five repetitions of this small text file.

   Use `wc` to count the number of characters and words in the original file and in the one you made from it. Explain the result.

   Create a subdirectory and move the two files into it.

3. Create a file containing a directory listing of both your home directory and the directory `/bin`.
4. Devise a single command line that displays the number of users currently logged onto your system.
5. Write, compile, and execute a C program that prints a welcoming message of your choice.
6. Write, compile, and execute a C program that prints its arguments.
7. Using `getchar()` write a program that counts the number of words, lines, and characters in its input.
8. Create a file containing a C function that prints the message `"hello, world"`. Create a separate file containing the main program which calls this function. Compile and link the resulting program, calling it `hw`.
9. Look up the entries for the following topics in your system's manual; the `cat` command, the `printf` function, and the `write` system call.
10. *You Must Try This One* There are two parts to the problem. If you are not able to do the first part of this problem, you are not prepared to take this class. If you find the second part extremely tricky, you will find the assignments for the course difficult and potentially more time consuming than you expect.

    part 1 Write a program that prints a range of lines from a text file. The program should take command line arguments of the form:

    ```
        lrange 10 20 filename
    or  lrange 10 20
    ```

    will print lines 10 through 20 of the named file. If there is no named file, the program should print lines read from standard input. If there are not enough lines in the file, the program should print what it can.

    Your program should work for input with any number of lines.
    Your program should work for lines of any length.

    part 2 Write a program called `last10` that prints the last ten lines of a text file. The program can be used from the command line with:

    `last10 filename` or
    `last10`

    If there is no filename, last10 processes standard input.
    Your program should work for input with any number of lines.

continued →

11. *Structs and Pointers* You must try this one, too. Write a function that computes some basic statistics for a list of numbers and stores those results in parts of a struct. In particular, given this definition:

```
struct numlist {  float *list;   /* points to list of numbers */
                  int   len;     /* number of items in list   */
                  float min,     /* the minimal value in list */
                        max,     /* the maximal value in list */
                        avg;     /* the mean of the numbers   */
               };
```

    write a function `compute_stats(struct numlist *listptr)` that takes as an argument a pointer to a struct numlist with *list* and *len* already initialized and computes and fills in the other three members. Write a program that uses your function to process user input and display results.

12. *Dynamic memory management*: Write a C program that reads in an arbitrary number of lines then prints those lines in reverse order. The lengths of the lines may be limited to a fixed maximum. but the number of lines is limited only by system memory available.

    For a slightly greater challenge, support a **-b** command line option that causes each line to be printed out backwards.

    The program, like most Unix tools, reads from a file if a filename is given or from standard input if no filename is given on the command line.

    Running this program on its own output should produce the original input. For example

```
./reverse /etc/passwd | ./reverse > rev2
diff rev2 /etc/passwd
```

    should show no differences.