Homework #2: The Static Quo

Issued: Tuesday, September 15
Due: Thursday, October 1

Purpose

This assignment allows you to learn more about Unix/C development, by improving your solution to the last assignment. You'll employ conventional modularity techniques, decomposing your one-file solution into multi-file modules. You'll enhance the capabilities of your solution, by counting input-file character occurrences for multiple sets of characters, not just vowels. We'll call a character set a *character category*.

Example categories are: vowel, consonant, letter, digit, punctuation, ascender, descender, upper, lower, curvy, and sticky. Categories can overlap.

We can't expect to predict all useful categories, so we should allow a user to name and specify categories. We could do this with command-line arguments, or a configuration file. For this assignment, though, categories will remain hardcoded, but in an easy-to-change way. Thus, category changes still require recompilation. Ugh! We'll fix this, eventually.

Assignment

Using your vowel-counting program as a foundation, write a C program that reads text from `stdin` and writes the character-category counts to `stdout`. Each line of output is a character-category count: the category name (e.g., vowel) followed by the number of input characters in that category.

In keeping with the "evolutionary" theme of our programs, you *must* represent your character categories with these module-local type and variable definitions:

```
typedef struct {...} ChrCat;
typedef ChrCat ChrCats[...];
static ChrCats chrcats={..., {0}};
```

Of course, you must replace the ellipses with real code. Memory for `chrcats` is allocated statically, rather than on the stack (`auto`) or in the heap (`malloc`). Whence this assignment's title.

Other Requirements

- Employ good modularity, formatting, and documentation.
- Do *not* use `<strings.h>` or its cousins. Write your own functions.
- In particular, encapsulate your character-category code in a separate module. Follow the example we saw in lecture:

  `pub/ModulePublic`

  Expose a minimal public interface, hiding as much as possible.
- Part of your module's interface should be a Java-like `toString` function. It should return a string representation of the character-category counts, suitable for writing to `stdout`. Use `asprintf` to construct the string, and `free` to deallocate it. A recursive `toString` function is quite elegant.
- Now, heed uppercase/lowercase differences.
- Aside from type representations, impose no arbitrary limits or sizes.
- Use this makefile:

  `pub/GNUmakefile`
- Run `valgrind` on your program.
- Demonstrate that you used a debugger to fix a bug.