Homework #5: The Object of My Affection

Issued: Thursday, November 5
Due: Thursday, November 19

Purpose

This assignment asks you, again, to improve your solution to the previous one. We will make several improvements, with an eye toward object orientation.

An array created by `malloc`, or one of its cousins, can be sized dynamically, at run time, but it is still a monolithic data structure. Unless you need fast indexing, enabled by adjacent elements, an array is a rather inflexible data structure. We want a finer-grained structure, which can grow or shrink piecemeal: one of the discrete structures (viz., lists, trees, and graphs).

A linked implementation of a discrete structure is often better than an array. Elements are not adjacent. An element can contain a "link" to another. A link could be an array index, but is usually a pointer: the address of the first byte of the other element. When you view memory as just an array of bytes, where an index is the address, it's all the same. A linked structure is more flexible that an array, because pointers can be changed, without moving elements.

In addition, our character-category module actually provides two data types: category (singular) and categories (plural). They should be provided by separate modules.

Finally, suppose we want to compute character-category counts for multiple input files. We would want each input file to have its own categories data structure, an *object*, for its categories.

Assignment

Develop your solution to this assignment in two steps, in two subdirectories, named something like: `ccc.list` and `ccc.mods`. Submit them both, together.

a) List

Begin this assignment, by modifying your previous program to store categories in a list, rather than an array:

```
static List chrcats=0;
```

as we saw in lecture:

```
pub/List
pub/ListStudents
```

This does *not* change the interface of your character-category module.

b) Antistatic Modules

Now, *refactor* your current categories module (e.g., `chrcats`) into two modules (e.g., `chrcats` and `chrcat`). The former will use the latter.

Then, eliminate the module-local statically-allocated variable `chrcats`, altogether (finally!):

- Change each function that assigns a new value to `chrcats`, to return the new value, instead. Let the caller of that function store the new value.
- Change each function that otherwise accesses `chrcats`, to take the value as a parameter, instead. Let the caller of that function pass the value.
- A good name for the new parameter is `this`.

These function-signature changes necessarily change the interface of your character-category module, so your main module will need to change too.

Also, of course, you want to hide the representation:

```
typedef void *ChrCats;
```

Change your `main` to create two character-category data structures. Put builtin categories in one; command-line categories in the other.

Your `main` should now have two local (auto) variables, like:

```
ChrCats ccs1=...;
ChrCats ccs2=...;
```

In object-oriented terms, each is a reference/pointer to what is almost an object. It has instance variables, but no methods. However, if some instance variables were function pointers, it would have methods, too.

Objection!

Contemplate adding function pointers to the data structure, for methods. The interface of your categories module would need only one function declaration: that of the constructor. The interface would also need a type definition, exposing the method declarations. Instance variables would be private. We'll discuss this in lecture.

Other Requirements

- Employ good modularity, formatting, and documentation.
- Do *not* use `<strings.h>` or its cousins. Write your own functions.
- Use this makefile:

  `pub/GNUmakefile`

- Run `valgrind` on your program. Fix your leaks.