Stat 290: Lab 5
C

Lab Objectives

1. To practice programming with C
2. To learn to read data from a file
3. To learn to write data to a file

Assignment

There is nothing to be turned in for this lab.

Getting the Data

This lab uses two datasets you have seen before: the used Mercedes price dataset and the mental health patients dataset. Get the data from the class web site and remove all header info, including the variable names. The only lines left in the file should be the actual cases. In the code below, I have named these files `mercedes.dat` and `lunatics.dat`.

Reading in Data

Data is easily read in from either a fixed-format file or a file with variables separated by white space (spaces or tabs). The standard command for reading in from a file is `fscanf()`, which is in `stdio.h`. `fscanf()` takes a variable number of arguments. The first argument is a pointer to the file. The second argument is a string of one or more variables to read in, using the same format as `printf()`. The remaining arguments are pointers to the addresses of each variable to be read in. Note that `fscanf()` is just like `scanf()`, except that it has an additional argument which is the pointer to the file.

Pointers to files

The first argument to `fscanf()` is a pointer to a file. Such a pointer must be declared as a pointer to type `FILE`. The pointer is then set to the right address with the `fopen()` function (also in `stdio.h`, which takes two arguments: the name of the file (as a string) and the method of opening (`r` for reading, `w` for writing (which will erase a pre-existing file of the same name), `a` for appending). A common default name for this file pointer is `fp`, standing for "file pointer". Since it is a pointer, no `&` is necessary in the `fscanf()` call. After you are done reading from (or writing to) the file, you should close the file with `fclose()`, which takes one argument – the file pointer. Here is an example of reading in a file called ``` ``data.dat'' ``` which consists of ten lines, with two integers on each line. A further description of the other arguments of `fscanf()` is in the next subsection.

```
main(){

  FILE *fp;
  int i;
  int a[10],b[10];

  fp = fopen("data.dat","r");

  for(i=0; i<10; i++) {
    fscanf(fp, "%d %d", &a[i], &b[i]);
  }

  fclose(fp);
}
```

fscanf()

The second argument of `fscanf()` is a string (denoted by double quotes) which gives the formats of the variables to be read in. If the file is a white-space separated file, then the widths of the fields does not have to be specified, as in the above example. If the file is a fixed-format file without spaces separating variables, then the widths of the variables must be specified with the standard `printf` formats, i.e. a six-column integer is `%6d`. Note that no `\n` is necessary, since a carriage return also counts as white space. Finally, the remaining arguments are pointers to the addresses of the variables in which to store the data being read in, which explains the necessity of the ampersands.

Writing Data to a File

Writing to a file is done with `fprintf()`. Just like `fscanf()`, `fprintf()` is very similar to `printf()`, except that it has an additional argument (which comes first) for a pointer to the file to which to write. Again, the pointer is assigned with `fopen()`, although now you use ``` ``w'' ``` or ``` ``a'' ``` as the mode (to write, erasing any pre-existing file, or to append, adding to any pre-existing file or creating a new one if it doesn't already exist).

Mercedes Example

The program `carsexample.c` contains code to read in the Mercedes prices dataset, compute the average price for each model of car, and write these average prices to a file called `mercedes.out`. Note that it also prints to the screen the second case in the datafile so that you can verify that it is reading in the data correctly. Also note that since the datafile contains several variables that we don't care about (the case number, age, mileage, and vendor), these variables are not completely stored – each new case overwrites the previous one. If we cared about these variables, we would have read them each into their own array. But since we don't care, we can save memory (not that it matters in this case) by overwriting them.

Compile and run this file. Check that the line printed to the screen does match the second line of the file. Check that the output file contains the correct averages (22078.8, 14846, 19157.5, 14883.3, and 9235.62).

Mental Health Patients Example

The program `lunatics.c` will read in the file `lunatics.dat` (which should have no header info) and just print it back out to the screen. Compile and run it to see that it works. There are several things to note about this code:

- The county name appears to be stored as a matrix of characters. Recall that arrays are like pointers, so matrices are like pointers to pointers. In either case, what happens is that each character of each name is stored in a separate element in this matrix, so that a row of the matrix contains a county name, letter by letter. We can process the whole name as a `string`. Recall that strings are terminated by a `\0` character that isn't normally displayed to the user. However, you must remember that it exists when allocating the array. Thus the longest name in the data is "Barnstable", which is 10 characters long. The variable must be allocated to have 11 columns to accommodate the termination character `\0`. Try changing this to 10 columns, re-compiling, running, and seeing what happens. Think about how the characters are stored to see why it does what it does. If you can't figure it out, ask in the next class.

- To read in a string, you can use `%s` in `fscanf()`. Bad things may happen if you haven't allocated a large enough string.

- To read in a float or a double, use `%lf`, which stands for "long float". Don't ask me why. It just works that way.

Finishing Up

There is no assignment for this lab. When you are done, you may want to clean up your directory to remove unnecessary output files (like `mercedes.out` or even unnecessary executables (note that executables take a lot more disk space than the `.c` files). You may also want to clean up any backup files that have been created (these will end with a tilde; on the ISDS machines, there is an alias `pu` that will do this for you).