COSC 301: Operating Systems
Project 1: Data scanner and listifier
Due: 25 September 2013
Fall 2013

1 Synopsis

Welcome to Project 1! There are two main objectives for this project: (1) to continue to improve your C programming skills, including more difficult concepts such as pointers, and (2) to acquaint you with "systems programming". To accomplish the former, you will write code that creates and manages a sorted linked list. To accomplish the latter, you will write this list-management code as a part of a program that does file I/O and uses functions in the standard C library that you are likely to encounter again.

2 Getting Started

To get started, fork the git repo at `https://github.com/jsommers/cosc301_proj01`. Clone the newly forked repo into your development environment.

Please note that there are some file skeletons set up to help get you started, as well as a working `Makefile`.

Also, you are welcome and highly encouraged to work with someone else on this project. You may also work in a group of 3, but note that I will expect especially strong work from such groups.

3 Detailed Description

You will be required to write a program named `proj01` that reads in a list of integers from a file or standard input, adds those integers to a linked list, and prints out some summary information.

You must write your code in two separate `.c` files: `main.c` and `list.c`. `list.c` will have all the functions associated with your sorted linked list, and `main.c` will have all the code associated with reading and processing the file (and using your linked list). Note that in addition to the list functions written in `list.c`, you'll also have to create the header file `list.h` which contains all the function and data type declarations necessary for `main.c` to compile correctly. Fear not: you'll be given some skeleton code in order to get you started.

All input data should be checked for correctness; any invalid data should simply be ignored. You should only accept *integers* from the input file — no floating point numbers (anything with an explicit decimal) or character strings. If you encounter bad input data, you should simply move on and read in the next data item as if nothing bad had happened. A sample data file is available in the git repo. (A suggestion: as you "tokenize" the input, pass each token into a boolean function you create, that tests whether the token is composed purely of integer.)

Each valid input data item should be added to a linked list. Your linked list code should insert each item in numeric order (*i.e.*, sorted).

The input data may either come from a file, or from standard input. If your program is started with an argument, as in: `./proj01 myfile.txt`, you should read input from the file `myfile.txt`. If your program is started *without* an argument, as in `./proj01`, you should read the input data from the special file `stdin`. There is already some code in the `main.c` starter file to help you set this up.

Once you've read all the input data, you should print out the list of integers you've read, followed by CPU resource statistics. Prior to your program exiting, you must clear and deallocate all items on the linked list.

For CPU resource statistics, you should print the amount of time your program spent executing in `user space` (*i.e.*, *not* in the operating system kernel), and the amount of time your program spent executing in the OS kernel. Please print these times nicely formatted. A sample of the kind of output your program should produce is as follows:

```
$ ./proj01 proj01-sample.dat
*** List Contents Begin ***
-65
12
23
25
33
34
34
68
99
100
10923
*** List Contents End ***
User time: 0.001885
System time: 0.008332
```

4 Useful C library calls

To read and parse the input data, and accomplish some other necessary tasks, you may wish to consult the following man pages:

- `fopen()` and `fclose()` are useful in opening and closing a file (i.e. the input data) Note that `fopen()` returns a `FILE *` type, and that `stdin` is a *predefined* `FILE *` that you can use to read from standard input (the keyboard). You do not have to open `stdin` — it is already available for you to use.
- `fgets()` is useful in reading in a line from an open file
- `strlen()` is useful for finding out the length of a C string
- `strtok()` is useful in breaking up an input string into whitespace-delimited "tokens"
- `isdigit()` is useful for checking if a character is a digit or not
- `atoi()` (or `strtol()`) is useful for converting a string into an integer
- `getrusage()` is useful for finding out the CPU resource usage of the current process
- `gettimeofday()` won't be used directly in this project, but the man page has information on the `struct timeval` structure used by `getrusage`. You'll have to know how to get information out of this structure in order to print the program summary information.
- `perror()` (or `strerror`) is useful for printing out the reason for a system call failure

5 Error checking and memory leaks

Part of your overall grade will be based on your program's robustness. This means that you'll have to ensure that you check for system call errors and handle them in some reasonable way. Each man page has a section titled *Return Values* which describes the values that can be returned and their meaning(s).

Part of your overall grade will also be based on how well you clean up your heap memory usage. Any memory that you allocate from the heap should be deallocated before your program exits. I recommend that you test your program using the `valgrind` tool to help debug any memory problems. In the simplest case, you can just run your program like:

```
$ valgrind ./proj01
```

There are a number of command-line options that you can provide to `valgrind` to force it to give you additional information. `man valgrind` for more information (or `valgrind --help`).

6 Submission

You should commit and push all your code up to github, and just post a link to your repo on Moodle. Be sure to indicate who worked on the code (you only need to make one submission per student pair).

Final note: review the lab assessment guidelines posted on Moodle before submitting. Make sure that your code works right *and* is well structured.