Document Version: 1.1
Assignment Date: 1/15/2015
Due Date: 1/29/2014, 11:59pm

Objectives:

- Warm up with C programming
- Practice with data structures
- Practice with dynamic memory allocation and pointers
- Practice with systems calls (i.e. related library functions).
- Learn some system calls
- Exercise reading the man pages in Unix
- Trace the systems calls that a program executes.
- Discipline your programs for black-box testing where the expected output will have quite rigit formats.

1 Project Description

Write a C program in Linux that will read from an input file a sequence of student records, will sort them, and will print the sorted records into an output file. The input file will be a text file containing information about one student per line. Each line will include the following information in the following format:

StudentID Firstname Lastname Department GPA

You can assume *Firstname*, *Lastname* and *Department* are single words each composed of ascii letters (lower + upper case). Assume *StudentID* is a 7 digit number between (and including) 1000000 and 9999999. Assume *GPA* is a floating point number between (and including) 0 and 4. There can be one or more space or TAB characters between the fields. You can use **`fscanf()`** function to read the fields of a line easily into your variables. Note that *Firstname*, *Lastname*, and *Department* can be any length, therefore make sure you do not make any assumption about their lengths and create the memory required to hold them dynamically (*Tip: the* ***m*** *modifier of* **`fscanf()`** *might be handy here*).

Your program will read the information from the input file and will build a *linked list* where each item (entry) of the list will contain information about one student. Hence, each item of the linked list will be a structure having fields to store information about a student (in C we do not have `class`, we have **`struct`**). You can build the list as a *doubly linked list*. In this case, each item in the list will have also a pointer to the next item and a pointer to the previous item. After building the list, your program will sort the list using the ***insertion sort*** algorithm. Sorting will be done based on the *StudentID* field. Then your program will write the list of items into an output text file. Each line of the output file will contain information about one student in the following format:

StudentID,Firstname,Lastname,Department,GPA

Note the comma (instead of space or TAB characters) between the fields. The output should contain no spaces or TABs, and no empty lines. So, after you write the last line of information, you should close the file. And this should be the end of your program. It is very important that you produce the output in this format since we will use this format in our automated tests.

2 Development

It is the requirement of this assignment that you have to use linked list data structure, dynamic memory allocation, and insertion sort algorithm. You cannot build the linked list in the sorted order while inserting the elements. You should build the list by simply inserting the nodes to the end of the list each time until the list is fully built. Then, you should implement a separate insertion sort function, and sort the linked list using this insertion sort function. Also, you cannot assume maximum number of characters for the strings (firstname, lastname, department), they can be any size and the memory for them should be created dynamically as well. In this way, you will be able to practice with malloc, pointers and structures.

The name of your executable file has to be **`uoflinsort`**. A sample invocation of your program can be like the following:

```
uoflinsort in.txt out.txt
```

Here, `in.txt` is the input text file, and `out.txt` is the output text file to be created by your program. It is very important that you follow the specifications.

Example 1 *An example input file (for example, `in.txt`) can be like the following:*

```
2040003 AAAA BBBBBBBBB ComputerScience 3.45
2040002  AAA     CCC   ElectricalEngineering  3.01
2040005  AAAAAAAAAAAAAAAAA BBB  ComputerScience 3.60
```

Then the output file (for example, `out.txt`) will be the following:

```
2040002,AAA,CCC,ElectricalEngineering,3.01
2040003,AAAA,BBBBBBBBB,ComputerScience,3.45
2040005,AAAAAAAAAAAAAAAAA,BBB,ComputerScience,3.60
```

3 Development

You will develop your program in a Unix environment using the C programming language. You can develop your program using a text editor (emacs, vi, gedit etc.) or an Integrated Development Environment available in Linux. *gcc* will be used as the compiler. You will be provided a Makefile and your program should compile without any errors/warnings using this Makefile. Black-box testing will be applied. Your program's output will be compared to the correct output. A black-box testing script will be provided on the class website, make sure that your program produces the success message in that test. A more complicated test (possibly more then one test) will be applied to grade your program.

4 Tracing the System Calls

After finishing your program, you will trace the execution of your program and write down all different system calls it makes. To do this, you can use the `strace` command available in Linux. See `man strace` for more details.

5 Checking the Memory Leaks

You will need to make dynamic memory allocation. If you do not deallocate the memory that you allocated previously using **`free()`**, it means that your program has memory leaks. To receive full credit, your program should be memory-leak free. You can use `valgrind` to check the memory-leaks in your program. `valgrind` will output:

"All heap blocks were freed - no leaks are possible"

if your program is memory-leak free.

6 Submission

Submission will be done through Blackboard strictly following the instructions below. Your work will be penalized 5 points out of 100 if the submission instructions are not followed. Memory leaks and compilation warnings will also be penalized if any. You can check the compilation warnings using the **-Wall** flag of gcc.

6.1 What to Submit

1. `README.txt`: It should include your name, ID, and list of system calls that your program made. Only the names of the system calls should be written in an alphabetical order where each line has a single system call name.
2. `uoflinsort.c`: Source code of your program.

6.2 How to Submit

1. Create a directory and name it as your UofL ID number. For example, if a student's ID is 1234567, then the name of the directory will be 1234567.
2. Put all the files to be submitted (only the ones asked in the *What to Submit* section above) into this directory.
3. Zip the directory. As a result, you will have the file 1234567.zip.
4. Upload the file 1234567.zip to Blackboard using the "Attach File" option. You do not need to write anything to the "Submission" and "Comments" sections. NO LATE SUBMISSIONS WILL BE GRADED!
5. You are allowed to make multiple attempts of submission but only the latest attempt submitted before the deadline will be graded.

7 Grading

Grading of your program will be done by an automated process. Your programs will also be passed through a copy-checker program which will examine the source codes if they are original or copied. We will also examine your source file(s) manually.

8 Changes

- Version 1.1: More clarification is added to the *Development* section.