Lab 2

Due No Due Date **Points** 15 **Submitting** a file upload
Available after Apr 3, 2021 at 12am

Working with File Input and Output

(3 pts) Design

In this lab, you will read and write to a file, as well as sorting the information. Copy and paste the following into a file named "inputs.txt":

```
7
199922007 C.J. Cregg Political_Science
200822013 Olivia Dunham Criminal_Justice
199822007 Josh Lyman Law
199922006 Toby Ziegler Communications
200922015 Leslie Knope Public_and_Environmental_Affairs
199922004 Sam Seaborn Law
200722013 Walter Bishop General_Sciences
```

▶ nput file provides details for a student database in the following format:

```
Number_of_students
ID_Number Student_First_Name Student_Last_Name Major
 …<repeats n number of times>…
ID_Number Student_First_Name Student_Last_Name Major
```

Your program will read specific information from the file and continue reading the contents of the body from the file until the EOF (end of file) character. You will write the following information to an output file:

- Sort students by ID number
- Sort students by last name

Each section of information should be labeled in the output file in all capital letters. A struct should be used to store and manipulate the file information between reading and writing the file. You must include the follow three functions with the exact prototypes:

```
student* create_student_db(int);

void get_student_db_info(student *, int, fstream &);

void delete_student_db_info(student *);
```

Your main function needs to check to make sure the file you open exists before moving forward. If the file doesn't exist, then you need to provide an error message and prompt the user to enter a file name that does exist.

- Write a design for the main function in the driver file, driver.cpp.
- Write a design for the create_student_db(), get_student_db_info(), and delete_student_db_info() as well as the functions needed to satisfy the above bulleted output functions in the implementation file, student_db.cpp

fstream documentation: **http://www.cplusplus.com/reference/fstream/fstream/ (http://www.cplusplus.com/reference/fstream/fstream/)**

(7 pts) Implementation

Now, implement the driver.cpp, student_db.cpp, and student_db.h files. Create a Makefile to manage the compilation of all these files. You can adapt the Makefile that was posted on the Lectures page in Canvas.

A Note on Sorting

There are many different algorithms that can be used to sort data. For the purposes of this lab, one of the simplest algorithms to implement is a version of the bubble sort algorithm. Bubble sort works by repeatedly comparing two elements in a collection of data, and swapping them. It continues through all of the data until no more swaps are needed, indicating that the data is sorted.

▶ ble Sort reference: **https://www.geeksforgeeks.org/bubble-sort/ (https://www.geeksforgeeks.org/bubble-sort/)**

(3 pts) Makefiles

What if we had 1,000 implementation (.cpp) files? Manually compiling all of them together would take forever and have a high chance for error. Luckily, UNIX/Linux has a built in script that makes compiling multiple files together easy called a Makefile. Just type

```
vim Makefile
```

on the command line to create it. Now modify the file following the pattern shown below:

```
<target>:
     <compiler> <file1.cpp> <file2.cpp> -o <target>
```

Note that the leading whitespace MUST be a tab (you can't just use spaces). An example of what this could look like for this lab would be:

```
student_db:
     g++ student_db.cpp driver.cpp -o student_db
```

Save and exit the file. You can type "make" in the terminal to run it.

Important note: If you copy and paste from the lab, your editor may convert the tab character to 4 space characters. You have to ensure your editor setting inputs a tab character! Also be sure that the "dash" character (which looks like this: -) is copied correctly.

One of the other benefits of makefiles is that you can add variables to it and make compiling happen in different stages by stopping g++ after compiling and before running it through the linker. This creates object files (.o files), which you can link together

```
CC = g++
exe_file = student_db
$(exe_file): student_db.o driver.o
     $(CC) student_db.o driver.o -o $(exe_file)
student_db.o: student_db.cpp
     $(CC) -c student_db.cpp
driver.o: driver.cpp
     $(CC) -c driver.cpp
```

Try to make your program again. Notice all the stages. In addition, we usually add a target for cleaning up our directory:

```
clean:
     rm –f *.out *.o $(exe_file)
```

Now we can run the specific target by typing "make <target>.

```
make clean
```

Makefiles are a useful way to automate and control the program building process as your projects grow in size.

(2 pts) TAR Files

When you begin to have projects with multiple files, it becomes useful to have a way to group them together for uploading, emailing, or archiving. TAR files (also known as tarballs) are classic way to do this on Linux machines, and are what you will use for programming assignment submissions.

To create a tar file, use the tar command. There are multiple options that can be added, with examples and explanations at the reference material below. The ones included in the example below are -c, -v, and -f.

```
tar -cvf tarfile_name.tar file1 file2 ... filen
```

The -c option is used to create the tar file. The -f option is used to specify which tar file to use, followed by the filename (ending in .tar) that you specify. The -v option stands for "verbose", and prints the files being added to the tar file to the terminal as the tar command executes.

After the command, options, and chosen .tar filename, you must include all of the files that you want to be added, separated by a space.

To extract the files from a tarball, you can use the following command, where the -x option stands for "extract":

```
tar -xvf tarfile_name.tar
```

Prove to the TA that you can make a tarball with multiple files, and then extract the files from it. Make sure that when creating the tarball you specify the .tar filename, and to extract the files into a different directory than the original.

TAR Command Examples reference: **https://www.rootusers.com/23-tar-command-examples-for-linux/ (https://www.rootusers.com/23-tar-command-examples-for-linux/)**