This lab will be done individually. You may discuss it with classmates, subject to the empty hands policy (see the course syllabus). As this is the first assignment in this class that requires you to write code, you should begin by reading Kelly Shaw's "Coding Standards" document on the "useful links" page of our class webpage. For now, you can ignore item number 8 in that document.

Assignment: In this lab, you will write your first meaningful C++ program that uses C++ vectors, and that reads and writes files. For this lab, you may work on any device you wish, but your final work must compile and run on the CS department Linux cluster (machines `mathcs01` - `mathcs06` or `turing2`).

Specifications:

- Your program must be named `ReadWrite.cpp`.
- You must accept as command-line arguments the names of the input file and output file, in that order. You must check for the appropriate number of arguments, and, if not correct, you must print a *useful* help message and exit before any crash or unexpected behavior occurs.
- You must use an `ifstream` stream and appropriate logic (use the slides from lecture and provided example program as a guide) to read from the input file. Be sure to check that the file opened correctly, and be sure to close the file when you're finished reading.
- Each line will contain a first name (as a string, with no spaces), a last name, a student ID (as an integer), and the year of study (as an integer), each separated by white space. Your program must work correctly whether there are zero lines, one line, ...
- You must use multiple `std::vector` instances to store the input values as you read them in.
- You must use an `ofstream` stream and appropriate logic (use the slides from lecture as a guide) to write an output file. Be sure to check that the output file opened correctly, and be sure to close the file when you're finished writing.
- The first line of the output file must contain all first names read from the input file, in reverse of the order they were read, each separated by a tab (`'\t'`). The second line of the output file must contain all last names read from the input file, in reverse of the order they were read, each separated by a tab (`'\t'`). The third line of the output file must contain all IDs read from the input file, in reverse of the order they were read, each separated by a tab (`'\t'`). The fourth line of the output file must contain all years of study read from the input file, in reverse of the order they were read, each separated by a tab (`'\t'`).
- Do not hard-code any filenames.
- Include a comment block at the top of your program containing your name, the current date, and a description of what the program accomplishes. Also include sufficient commenting throughout your program, consistent with the coding standards document given on the course web page. Use consistent and appropriate indentation and white-space.
- Include a plain-text (not Word, RTF, PDF) file named `README.txt` that describes the test cases you used to convince yourself of correctness.

Creating, extracting, and viewing tar files: Like most unix commands, there are many flavors. I'll give you just one of each (though you are welcome to explore the man pages to learn and use more). To create a tar (short for "tape archive") file called `foo.tar`, and that contains the files file1.c and file2.c, you should move to the directory that contains both files (I assume they are both in the same directory). Once in that directory, you can create the tar file by running the command:

```
tar -cvf foo.tar file1.c file2.c
```

The flags here stand for the following: 'c' means create a tar file, 'v' means do it in verbose mode (list the files being added to the tar file), and 'f' means write (or in other commands, possibly read) the files to the specified tar file name.

If you have a project directory, perhaps called `project1`, and you wish to create a tar file that maintains the directory structure of your project, move to the parent directory of `project1`, and run the command

```
tar -cvf foo.tar project1/*
```

This will tar everything in the project 1 directory. Moreover, when you extract this tar file, it will create a directory called `project1` which will be an exact copy of your project1 directory.

To extract the tar file `foo.tar`, run the command

```
tar -xvf foo.tar
```

The 'x' flag here is short for "extract".

Finally, to view the contents of a tar file without extracting it, use the command

```
tar -tvf foo.tar
```

The 't' flag here means you want to examine the contents of the tar file.

Grading Standards: I will specifically be looking for the following. Make sure that, at least, your program covers these.

- Your program must compile without warnings, and execute to receive credit.
- Your program must fail gracefully given an incorrect number of command line arguments. The printed message must be useful for a novice user of your program.
- Your program must handle an empty input file appropriately. The resulting output file should also be empty.
- Your program must handle one-line, short multiple-line, and long multiple-line input files appropriately.
- Your program contains a block comment per specifications.
- Your program contains reasonable inline commenting.
- Your program uses proper whitespace, consistent indentation, and good choices for variable names.

Naming:

Because everyone will be submitting their lab to a single Box folder, they can't all be called "Lab1.zip" or the like. Instead, they must have some indication of who completed the submission. Naming is also important because many of these labs are graded using software that expects specific naming conventions. For this lab, your submission **MUST** be named `cmsc_240_lab1_`*netid*`.zip` or `cmsc_240_lab1_`*netid*`.tar`, where *netid* is, of course, your netID. I thank you in advance for your cooperation.

Submission:

The high level picture is that to submit any labs/project in this course, you send an email to a project specific (i.e., unique to the project) email address with your **single** submission file attached. This has the effect of placing your submission in the appropriate Box folder. If your submission requires more than one file, you should `tar` or `zip` your files together to create your single submission file (see instructions above).

The the email address for this lab is `lab1.eys7vuaxm9c2f5bx@u.box.com`. Thus to submit this lab, you should attach your tar or zip file to an email sent to the email address `lab1.eys7vuaxm9c2f5bx@u.box.com`.