In this assignment, you will install the development environment that we will use for programming in this course, and you will write a small program to make sure the environment is installed properly. This assignment is to be completed individually.

First, install the development environment for the course. The installation is quite complex. Please be sure you allow enough time to complete installation and to write the program for this project. The document attached to this project contains information on how to install VirtualBox, Lubuntu, and CodeLite. Please follow the instructions in the document carefully so that you do not omit any steps. You may post any questions you have to the “Questions about Project 1” discussion board in Canvas.

Once you have installed the development environment, write a simple program that reverses the lines in a file. The program should read in the data from the specified input file and reverse it. The lines should be printed out in the reverse order of the input stream. For example, if the input file contains the following text:

```
A Time to Talk

When a friend calls to me from the road
And slows his horse to a meaning walk,
I don’t stand still and look around
On all the hills I haven’t hoed,
And shout from where I am, ‘What is it?’
No, not as there is a time talk.
I thrust my hoe in the mellow ground,
Blade-end up and five feet tall,
And plod: I go up to the stone wall
For a friendly visit.

— Robert Frost
```

the output (either to a file or to the standard output (the screen) depending on if an output file name is provided or not) would be:

```
— Robert Frost

For a friendly visit.
And plod: I go up to the stone wall
Blade-end up and five feet tall,
I thrust my hoe in the mellow ground,
No, not as there is a time talk.
And shout from where I am, ‘What is it?’
On all the hills I haven’t hoed,
I don’t stand still and look around
And slows his horse to a meaning walk,
When a friend calls to me from the road

A Time to Talk
```

The program must accept either one or two command line arguments. The first command line argument will be the name of an input file. The second command line argument, if present, will be the name of an output file. <u>If the name of an output file is provided as a command line argument, the output will be written to the output file only. If the name of an output file is not given as a command line argument, the output will be written to standard output (the screen)</u>.

Setup

In CodeLite on your virtual machine:

- Create a new C++ project named Project1. See the installation document for information on how to create and configure your CodeLite project.
- For this project, you may write all of your code in the main function in a file called main.cpp.

Submitting Your Project

Compress (zip) the src folder of your CodeLite project. You will need to do this using your operating system's file manager because CodeLite does not provide a way to zip the source folder. (It's probably easier to do this on your host computer. This will work as long as your workspace is in the shared folder.) Name the zip file YourLastNameYourFirstNameProject1. Upload this zip file to Canvas before the due date. Remember that late assignments are not accepted in this course. See the syllabus for details.

Implementation Notes

Command Line Arguments

You program must accept either one or two command line arguments. The first command line argument is the name of the input file. The second command line argument, if present, is the name of the output file. (Remember that the value in argv[0] is the name of the program.)

<u>If the name of an output file is provided, your output must be written only to that file. If a name of an output file is not provided (only one command line argument), your output must be written to standard output (the screen).</u>

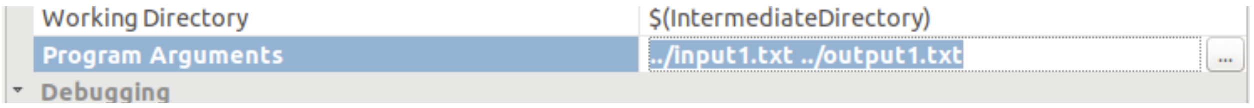
If the number of command line arguments is incorrect you should print an error message to the *cerr* stream and exit the program with a return code 1. Your message should be in this format:

```
Usage:  ./Project1 <input filename>  [output filename]
```

- Note that `./Project1` is the name of the program and is found in argv[0]. Angle brackets are traditionally used to indicate required arguments and square brackets indicate optional arguments.

CodeLite

- To add command line arguments to CodeLite, right-click your project and choose "Settings." In the "General" tab under the "Execution" category, you will see "Program Arguments." When you select "Program Arguments" a button with three dots will appear. Click this button to enter your arguments.
- Please place your data files in your Project1 directory. You may then refer to a file in your code using the path: ../<input filename>
- Here is an example in CodeLite where both an input file and an output file are provided as command line arguments:



Working Directory	$(IntermediateDirectory)
Program Arguments	../input1.txt ../output1.txt

Useful types and functions

- Use the C++ type ifstream (https://en.cppreference.com/w/cpp/io/basic_ifstream) for the input file.
- Use the C++ type ofstream (https://en.cppreference.com/w/cpp/io/basic_ofstream) for the output file.
- Use the function getline (https://en.cppreference.com/w/cpp/string/basic_string/getline) to read each line of the file into a std::string type (https://en.cppreference.com/w/cpp/string/basic_string)
    - The getline function returns a value which can be tested as true or false. A false value indicates end of file or error.

Error Handling and Assumptions

- If the input file and the output file have the same name you should print an error message to *cerr* and exit the program with a return code 1. Since you are writing all code in the main function you may exit the programming with the statement `return 1`.
- If you cannot open a file, you should print an error message to *cerr* and exit the program.
- You may not assume anything about the length of the input file. It might be very long. Be sure to choose an appropriate data structure.
- Be sure to close any files you have opened before your program ends.

Grading Criteria (10 points possible)
Note: Your program must compile to receive credit for this assignment!

Points	Criteria
0-2 points	**Command Line Arguments:** Does the main function accept one or two command line arguments of type std::string? Is a “usage” error message printed if the number of command lines arguments is not correct?
0-3 points	**Input:** Does the main function correctly read lines from the input file and store the lines in a data structure? Is the program able to read a file of any length?
0-2 points	**Output:** If the name of an output file is provided, are the lines of the input stream printed to the output file only (not standard output) in reverse order?
0-2 points	**Output:** If the name of an output file is not provided, are the lines of the input stream printed to standard output (the screen) in reverse order?
0-1 point	**Style:** Is the code easy to read? Does it follow class style guidelines (See Program Style page in Canvas.)