Laboratory 12 Activity Instructions

1. Download the **`Lab12Activity.cpp`** file from WebCampus, and download the **`testfile.txt`** file. This program will need some lines of code filled in, but will be compilable and will work in a limited way as soon as you build and run it. In addition, open the **`testfile.txt`** file in your Dev C++ editor. This file has two lines in it. One that shows some text and an integer, and the second one is just text. If you copy the text from WebCampus and create your own file, make sure you have an extra empty line after the second line.
2. You will notice on lines 39 and 50, the function **`getline`** is used. This function accepts all text including spaces and endlines up to either a ***one less*** than character limit (e.g., **`MAX_STR_LEN`**), or until it finds a specific character (e.g., **`COLON`**); it accepts one less than the character limit so that a **`NULL`** character can be added to the end of the string. When the function has completed, the input characters will be loaded into the string parameter (e.g., **`testStr`**) from either a file stream (i.e., **`fstream`** objects) or a console input stream (i.e., **`iostream`** objects). If for example you wish to load text with spaces such as a person's full name or address, you can do this from a file or from the keyboard. From this description, you should be able to see that the **`getline`** operation on line 39 will capture all the text up to and including the colon in the file, and the **`getline`** function on line 50 will capture all the text on the second line of the file. Go ahead and run the program.
3. You will notice that the second string was not displayed. This is because after the integer that was input in the extraction operation (i.e., **`inf >> testInt;`**) only the integer was input, but the endline character following the integer was left in the data stream. Thus, the next **`getline`** operation immediately found the endline character and stopped before it took in any characters.
4. Any time you have to combine **`getline`** and extraction operations, you will need to resolve the issue that an endline character may be left in the input data stream. This is not a big problem, but it must be considered. The solution is to place another **`getline`** in the program after the integer input and before the next string input. Place the code **`inf.getline( testStr, MAX_STR_LEN, ENDLINE_CHAR );`** at line 47 of your editor. This will resolve the issue; build and run your program to verify.
5. It seems kind of inefficient to input dummy data into a string just to get rid of an endline character however. So, a simpler function called **`ignore`** can be used. This function takes the same parameters as the **`getline`** function except it doesn't need the string to store the unused endline character. Replace the **`getline`** code on line 47 with **`inf.ignore( MAX_STR_LEN, ENDLINE_CHAR );`**. You will again find that this also makes your program work correctly without having to use a string.
6. The **`getline`** and **`ignore`** functions are handy and easy to use for assisting with input of data having spaces, which the extraction operations cannot handle. However, wouldn't it be more interesting if you could write your own **`getline`** and **`ignore`** functions?
7. These functions are called member functions of the **`iostream`** and **`fstream`** classes, so they work with the dot operator (i.e., it is used as **`fin.getline(…);`** or **`fin.ignore(…);).`** You cannot add member functions to these classes, so you will have to pass the **`ifstream`** parameter into the function along with the others. That will be the only difference.
8. The stub functions for the **`inputGetLine`** and **`inputIgnore`** are created for you with the correct parameters and the step five comments.

9. You will need to use the member function **`get`** to accomplish your tasks. You can refer to **`cplusplus.com`** for this, but it is pretty simple. The function **`get`** acquires one character from the stream and returns it as an integer. Unlike the extraction operator, the **`get`** function acquires ***every*** character including the endline character and spaces and tabs (i.e., white space). This way, you can input string data with spaces ***and*** you can test for an endline character as you iterate through inputting the data. The **`get`** function is used as follows:

```
inputCharacter = char( fin.get() );
```

10. Note that you must cast the return from **`get`** in order to convert it from an integer to a character.
11. Write the functions **`inputGetLine`** and **`inputIgnore`** and then test both of them on line 47. When they work correctly on line 47, try the **`inputGetLine`** function on lines 39 and 50.

If you do not complete this in class, it is worth your effort to finish this program on your own.