Laboratory 9 Activity Instructions

1. Download or copy the **`Lab9Activity.cpp`** file from WebCampus. Implement the following activities, but compile and run the program after each step

2. You will need an **`fstream`** include file for the file operations and a **`cmath`** include file for the random number generator. Add these at lines 3 and 4 in the source file.

3. The following exercise assumes that your line numbering is turned on in your editor. If it is not, you will need to go to "Tools", "Editor Options", then go to the "Display" tab, and then select the "Line Numbers" box. The line numbers for the following are assumed to start at line 1 so you are better off not to check the "Start at Zero" box.

4. At line 41 is the function **`generateNumbers`** that generates a series of random numbers and writes them to a file. The number of numbers is specified by the user. At line 44, the **`displayNumbers`** function shows the numbers from the file, but returns **`false`** if the file was not accessed or no data was found. Notice how this is used as both an action function and test function.

5. Carefully review the following conditions for the **`generateNumbers`** function:

    a. At line 71 in the **`generateNumbers`** function, note that when a file is opened for OUTPUT, no clearing or testing needs to be implemented. The loop at line 76 iterates exactly the number of times specified by the user (and passed to this function as a parameter). Next, the random number generator provides a value at line 79.

    b. The test for **`counter > 0`** on line 82 provides a comma and a space BEFORE every output value except the first. This keeps the first one from having a comma/space prior to it, and it keeps the last output value from having a comma/space after it.

    c. At line 100, a final end of line is added. If there is not an end of line after the last item in a file, the last item will not be accessible for input later on; the file input operation MUST have some white space after every value that is to be input.

    d. The modulo math on line 89 forces an end of line so that the number of numbers per line is managed.

6. Now implement the following actions for the displayNumbers function:

    Place **`inf.clear();`** on line 115 and **`inf.open( FILE_NAME.c_str() );`** on line 116; **`clear`** should be implemented before **`open`** on all INPUT file opening operations

    Place **`inf >> value;`** on line 119. Any time you are inputting data from an unknown number of items in a file, you MUST prime your loop with an advance read operation

Place **`while( inf.good() )`** on line 122. This will test to make sure the most recently attempted input operation was a success every time the loop iterates

Once you have verified with the loop control that data was actually available, you can input the comma that is expected to be after every value. Place **`inf >> dummy;`** on line 125.

Place **`if( itemCounter % ITEMS_PER_LINE == 0 )`** on line 128. This will limit the number of items to be placed on each line

At line 148, place **`inf >> value;`**. This will attempt to reach for the next data item if it is available

On line 158, replace **`return false;`** with **`return ( itemCounter > 0 );`**. This will report if the function actually found any data

7. Once you have added in all the required code items, make sure the program compiles properly and then run it. You should see the numbers displayed on your screen.

8. Next, go to **File/Open**, and in the **Open File** dialog box, select **All files (*.*)** in the **Files of type** text box. This will allow you to see the **`testfile.txt`** file. Select this file and open it. Make sure that it has the number of numbers your program requested. Note the comma delimiting, again, only after the first item and not after the last one. You can open and check any text file you have created in your Dev C++ editor as needed.

If you have any problems, ask your TA for help