

SY DE 121

Lab Number 8

You are expected to complete both of these exercises. However, submit only one (i.e., your "best") of the two exercises for grading purposes.

Exercise 1:

Learning Objectives: Opening and closing files. Using strings. Using arrays of structs. Sorting. Passing filestream objects. Reading from and writing to files.

Read This First

In this problem, you will be asked to count the number of occurrences of each lowercase letter found in a sentence. Just focus on determining the number of occurrences of each *lowercase* letter (ignore uppercase letters and all other alphanumerics – this makes writing the program a bit easier!).

Also, you are required to output the data in a sorted fashion. So, the letter with the most counts should appear first, then the next letter, etc. You are welcome to use the sort algorithm of your own choosing. For example, for the input string “I need some sleep!!!”, the output should appear in the following manner:

Letter	Occurrence
e	5
s	2
d	1
l	1
m	1
n	1
o	1
p	1

What to do

Set up your program to read in the line of characters from a file (you can create test *.txt files using Notepad). Choose a suitable input file name (e.g., input.txt). The output filename can be hardcoded i.e., force the output file to be “output.txt”.

As in Lab #7, set up your project using an interface file (lettercount.h), an implementation file (lettercount.cpp), and the main file (main_lettercount.cpp) (the naming is helpful when the TAs have to grade your submissions, so please use the indicated names). Your main program should perform the following tasks in this order: allow the user to enter the input filename, set the output file name, open the input and output file streams (with error checking), then call a function to perform the counting (pass both file streams to this function), and then close the file streams.

Make proper use of the ‘const’ parameter modifier when creating functions.

Note: You need to modularize your code. As an example, Prof. Clausi’s solution code has 8 functions defined in lettercount.h (other than the main function). You may not create exactly the same type or number of functions, but this gives you an idea of the extent of the modularization.

Hints

- The small letters in the alphabet (a to z) are listed sequentially in the ASCII character set (see Appendix of your text). The compiler will treat 'b' as being larger than 'a', 'c' as being larger than 'b', etc. You can use `int(char)` as a means of determining the ASCII numerical equivalent for a given char. Similarly, you can use `char(int)` as a means of determining the ASCII character for a given integer. The integer equivalent for ASCII letter ‘a’ is 97, which might be a helpful const in your program.
- The `getline` function can be used with a sentinel character other than the newline character. For example, to use a period '.' as a sentinel character for a `getline` call, do the following:

```
instream.getline( line, 1000, '.' );
```

where `line` will store a maximum of 1000 characters from the `instream` ifstream up to (but not including) the terminating '!'.

- If you are using the `string` class, note that the `getline` function is called in the following manner:

```
string line;  
getline( instream, line, '.' );
```

- If you are using the `string` class, you sometimes have to convert `string` objects into `cstrings`. To do this, use the `'c_str()'` member function. For example, the `'open'` member function is not overloaded to handle `string` objects (such as `infilename`), so do the following:

```
instream.open( infilename.c_str( ) );
```

- Instead of just sending your output to a file, you should also send it to the screen. This should assist code debugging. To create a separate function to send the output to the screen is appropriate.

What To Hand In

Upload your source codes to the Dropbox folder.

Exercise 2:

Learning Objectives: Opening and closing files. Using strings. Using multidimensional arrays. Reading from and writing to files.

Read This First

Basically, this exercise asks you to multiply two matrices and write the result to a file.

What To Do

Setup your program in the same manner as Exercise #1 ie. use the three different files (in this case, named 'matrixmath.h', 'matrixmath.cpp', and 'main_matrixmath.cpp') and allow the main program to perform the actions in the same manner (cutting and pasting should save some time here!).

Read the two matrices from the same file. Here is an example of such a file (asterisks are not included in the file). Read the first two numbers as the row/column of the first matrix and then read in the first matrix. Then, do the same thing for the second matrix. Assume that there are no blank lines in the data file. Assume that only integer arrays will be used.

```
*****
3 4
1 1 1 1
2 2 2 2
1 1 1 1
4 5
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
1 1 1 1 1
*****
```

Since you have not been taught dynamic memory allocation, just set up your matrices in your program to some fixed large size eg. 100. Make sure that you only perform operations on and display the "true" parts of the array ie. only output the resulting 3x5 matrix, not the 100x100 matrix. Set error conditions if the array bound is incorrect. Output the product matrix into an output file (and to the screen as well). Echoing the input data to the screen is quite helpful, to make sure that you read the data file properly into the matrices (write a separate function to display a single generic matrix given its rows and columns).

Do not forget to check that the matrix dimensions are appropriate for multiplication. Exit if they are incorrect.

Note that your TAs will probably use input files different than the one above, so you should write your code to handle any matrices (up to size 100x100, of course).

Make sure that you properly use 'const' modifiers wherever appropriate.

What To Hand In

Upload your source codes to the Dropbox folder.

Due Date

Due date for all materials is Friday, November 10 at 6:00pm.