1_File Input/Output (I/O)

(0) Create a new (console) project. Open/edit the (default starting) file main.cpp

(1) Sometimes, opening and reading from an input file can be downright frustrating. Often it is just a function of trying to figure out which directory your IDE requires. Note: each IDE is finicky as to where your input files must be stored relative to your executable. Here's a tip when you need to read from a file: **get a *very small* program to work first**. Yup, *ignore the problem you are trying to solve*, at least for now. Just see if you can do this:

(a) Move the input files to the correct location
(b) In your C++ file, build/concatenate the full file path and name; open the file
(c) check to see if it opened ok? If not, print a nasty message to the console.
(d) Read one line *only*!!
(e) Print that line to the console.
(f) Close the file.

```
#include <fstream>

ifstream FIN;

string fileName, fullPath;
cout << "Enter filename to open: ";
cin >> fileName;

fullPath = "1_FileIO/" + fn;  // concat folder and filename

FIN.open( fullPath.c_str() ); // open requires old-style C string

if ( FIN.is_open() )
{
   string oneLine;

   getline(FIN, oneLine);    # read one line from FIN stream
   cout << oneLine << endl;

   FIN.close();
}
else
   cout << “NASTY MESSAGE: “ << fullPath << “didn’t open!” << endl;
```

If you can get these steps to work, then you can return to the problem at hand. Make sure you can get this to work *before* you continue with the next steps.

(2) Write a C++ program to:

(a) Prompt the user to entire an input filename. Note: I have given you two files to use for testing, both in the folder(directory) called **`1_FileIO`**; the two files are: `1_FileIO/`**`1a_story.txt`** and `1_FileIO/`**`1b.story.txt`**.

(b) In `main()`, open a text file for **input**. Make sure to print an error message if the file does *not* open properly.

(c) In addition to opening a file for reading, also open a (new) file for **output** to hold a report. This file should be named the same as the input file but have an "**`.xls`**" file extension added to the end of the filename. To learn how to open a file for writing, google this: "***C++ open file for writing***".

(d) Assuming the two files open OK, read lines of text (**one line at a time** until reaching the end of the input).

(e) For each line of input, print **tab-delimited** information (see below) to your output (report) file. How to find the number of vowels? Remember, a C++ string is an object that holds an array of characters. You can ask a string object for its length. You can also peek at individual letters (with a loop), for example: someString[0], someString[7], …, someString[i], …

Since we want "tab delimited" output in the file, make a constant for a TAB character as shown here: **`const char TAB = '\t';`** Also remember to include a line of headings in your output file.

```
FOUT << "Line#" << TAB << "Number of chars" << TAB << "Number of vowels"  << endl;
```

Here is what a sample output to the external file might look like:

```
Line#     Number of chars         Number of vowels
  1           20                      7
  2           34                     21
  :            :                      :
```

(3) Close the two files.
(4) Once your program excecutes, locate and **double-click on your output file**; given the **.xls** file extension, this should open in Excel. Make an appropriate bar chart of your data; that is, the bar chart should show for each line the number of characters and the number of vowels on that line.

Call me over to see your bar chart when you have finished this part.

(5) Alter your program to determine:
  (a) the line# with the LARGEST number of characters
  (b) the line# with the LARGEST number of vowels

(6) Add messages with the "largest" values and associated line numbers to the bottom of your output report.

Call me over to see your bar chart and new "largest" output when you have finished this part.

`2_Arrays/`: "Pretty Maids All in a Row"

(7) The **`2_Arrays/`** folder contains a file: **0_arrayZERO.cpp**

(a) Create a new (console) project (or use the one you already have been using; of course, replace exisiting files with the new ones as you need them).

(b) Load **0_arrayZERO.cpp** into your project. Study the code in 0_arrayZERO.cpp. Draw a *neat(!)* picture of the array called **someArray** in the space below. Include subscripts.

(c) **Before you run it,** handtrace the code. What do you expect for output? (**Fill-in the left column only**)

Expected Output (from handtrace)	Actual Output

(d) **Run it** to verify your expected output. **Write down the Actual Output** in the right-hand side above. Review your handtrace if your Actual Output did *not* match your Expected Output.

---

(8) Load **1_arrayONE.cpp** into your project. Study the code; read the comments. Alter the program so it produces the correct desired output.

---

(9) Practice Exam Question: Review the array declaration below and draw a picture of B. Handtrace the following lines of code, that is, show the output.

```
short B[8],  i;

// Draw a COMPLETE picture of B below.
```

```
for (i = 0;  i < 8;  i++)
       B[i]  =  i % 2;

for (i=1;  i < 7;  i++)
       B[i]  =  B[i-1]  +  B[i+1];
```

// What is in B now? Show a new picture below. Call me over when you have your answer.