Inlab #7

This lab is worth approximately 0.5% of your course grade. Recall that you can drop two in-lab scores.

Get the file stringCount.cc from the class web page, and put it in your eng101 space. Open stringCount.cc with your favorite editor. Put your name, uniqname, section number, and date at the beginning of the file in the first comments section.

The program has a main and three functions. The main function inputs a filename as a string. One of the functions, countChar, counts the total number of characters in a file. You do not need to change anything in main or in the countChar function. Complete the other two functions, countLine and countFile. CountLine should count the number of lines in the file, while countFile should count the number of times the given character appears in the file. (15pts)

Strings are a data type with a lot of built-in functionality. They are declared much like any other variable. Strings can be "added" which concatenates (tacks) the strings together, and compared using <, <=, >, >= or ==. One also commonly wants to know how big a string is (how many characters it has) to avoid an index out of bounds, much like other arrays. As you have seen, C++ provides .size() for the length of the string, and .find() to search for matches within strings. There are a number of other functions built-in to the C++ string type to make working with strings easier.

There is another kind of string that you will run into, namely older C-style strings. There are two primary needs for C-style strings in C++. The first is for opening file streams with a string variable containing the file name. Standard C++ strings don't work here, directly. The string must be converted to a C-style string using the function c_str().

```
ifstream inFile;
inFile.open("data.txt");              // a fixed string constant works fine
string filename = "data.txt";
inFile.open(filename);                // this will not work
inFile.open(filename.c_str());        // need to do this
```

The other main need for C-style strings relates to the"odd" main definitions you've seen on occasion, and is used for something called command line arguments.

```
int main(int argc, char * argv[])
```

When you type "nedit lab6.cc" on the command line (linux prompt), the name "lab6.cc" is passed to the program "nedit" as an argument. C++ gets this information through the "argc" and "argv" variables, which stand for the "argument count" and the "argument values." The argv variable is an array of C-style strings containing the parts of the command line that were separated by spaces. The two parts were "nedit" and "lab6.cc" in this example. The command itself is always stored in argv[0]. The argc variable holds the size of the argv array.

Also notice the .fail() stuck on the end of the ifstream variable name inFile. This returns true if there has been some failure reading from that ifstream. The main failure of interest here is the end of the file, because we do not know ahead of time how big the file is.

Think about the following questions, and type answers in the second comments section at the beginning of your code. (15pts)

In order to verify that your program was done, you probably did one or more of the following:

- Did not do anything to verify that it worked
- Compiled it and saw that it has no compiler errors
- Ran it with a large input file and made sure some results are displayed even though the expected results were not known beforehand
- Checked that it was identical to a neighbor's code
- Ran it with a small input file for which the expected answers were known
- Asked a GSI to verify that it works
- Something else (describe if appropriate)

1. Which two of the above should you do to verify that your program is finished and works? Briefly, why?

2. This program counts "words" by counting the spaces between words. This approach has some flaws. It may give incorrect word counts. What are two types of cases where this approach would give incorrect word counts?

3. What does the program do if you tell it to examine a file that does not exist?

4. In main, the code checks if argc equals 1. Give one example of how would you run your program from the terminal in order for argc to not equal 1.

The countChar function as written uses two nested while loops. The outer while loop reads the file line-by-line, while the inner loop goes through the current line character-by-character, as the code on the left below shows. The same approach might be written like the code on the right.

5. What is different between the two code segments? Does the alternate code work correctly? What might be bad about the difference seen in the alternate code? (Hint: what if the inside loop had 100 lines instead of just 2?). Bonus: How might we write this code segment using only one while loop?

```
int i=0;
getline(inFile, line);
while(not inFile.fail())
{
        i=0;
        while (i<line.size())
        {
                count=count+1;
                i = i + 1;
        }
        getline(inFile, line);
}
```

```
int i=0;
getline(inFile, line);
while(not inFile.fail())
{
        while (i<line.size())
        {
                count=count+1;
                i = i + 1;
        }
        i=0;
        getline(inFile, line);
}
```

Print your completed program & replies with: **`enscript -G stringCount.cc`**
You are to turn in your code (with the answers) in your Wednesday or Thursday lab of next week (The 17th or 18th).