CSC 112 Lab 5
Character Count
Spring 2015

Due
3:00pm Friday, 2/27

1 Introduction

Understanding the frequency of characters used in a body of text can be useful for various purposes. For example given an encrypted text, character frequency analysis is often used to determine the type of encryption used. For this lab you will write a program to determine the character frequency of a text file.

1.1 Character Frequency

We are interested in determining the character frequency (a count of the number of times a character is used) of a text file. The program will first prompt the user for the name of the file. If the file does not exist, re-prompt until a legitimate file name is provided. Afterwards, read the file and process every character in the file. Once the the frequency has been determined print out the top 5 letters, top 5 numbers, and top 5 *other* characters used in the file, as seen below (character in brackets then the count). Also note, the total number of characters per type is displayed. Consider upper case and lower case letters as the same; therefore, convert uppercase letters to lowercase. The *other* category consists of all non-alphanumeric characters.

Terminal

```
> ./lab5
Enter the name of the text file -> roll.txt

alpha top 5 out of 26
---------------------
[e] 2324
[t] 1502
[a] 1470
[r] 1444
[o] 1408

digit top 5 out of 10
---------------------
[0] 457
[2] 312
[8] 202
[1] 185
[9] 52

other top 5 out of 28
---------------------
[ ] 4137
[.] 458
["] 236
[,] 207
[
] 189
```

2 Program Design

Although character frequency is a simple task to describe, there are several lists that must be properly managed. Your program **must** adhere to the following program design requirements.

2.1 A `struct` and Arrays for Character Counts

As characters are read from the file, you will keep track of the number of times a character appears in the file. Although we do know all the possible characters that could appear, your lists should only store the unique characters encountered. The character list should store two items per element, the character and the count. Use the following `struct` to store the two items.

```
#include <iostream>
#include <fstream>
// maximum string size
#define MAX_STRING_SIZE 20
// maximum character list size
#define MAX_LIST_SIZE 50
using namespace std ;

//=== data type to store a character and its count ================
struct CharFreq
{
   char ch;     ///< the character
   int  count;  ///< the number of times the character occurred
};
```

As seen above, the `struct` definition should be placed immediately after the `using namespace std;` statement in your program. This placement will allow it to be used everywhere in your program. For example, you should create an array for the alphabet (letter) characters as follows.

```
CharFreq alphaList[MAX_LIST_SIZE];  // list of unique letters
int alphaNum = 0;                   // number of unique letters
```

You should create similar lists for numbers (digits) and the other characters.

2.2 C-Strings and Reading the File Name

Strings are multiple characters in the form of a list, or array in C/C++. However, strings in C/C++ are considered special `char` arrays called "C-strings." What is special about C-strings compared to simple `char` arrays is that C-strings always end with the *null terminator* character. In C/C++ the null terminator is the character `'\0'`. *Why do we care?* We will read the file name from the user as a C-string. The following code will read characters from the keyboard until a space or newline is entered, or until the maximum string size is exceeded (`MAX_STRING_SIZE`). These characters will be stored in the array `fileName` and the null terminator will be added to the end.

```
void readFileName(char fileName[])
{
   int i = 0; ///< local counter
   char ch;   ///< a character from the keyboard
   cout << "Enter the name of the text file -> ";

   // read in characters until a space, newline, or out of room
   ch = cin.get();
   while(ch != '\n' && ch != ' ' && i < (MAX_STRING_SIZE - 1))
   {
      fileName[i] = ch;
      ch = cin.get();
      i++;
   }
   // add null terminator to the end to make it an official C-string
   fileName[i] = '\0';
}
```

2.3 Reading the File and Processing Characters

Reading the file and processing characters must be done in a separate function. The function will first attempt to open the file and if the file does not exist, then return `false` and read another file name from the

user (call the `readFileName` function above). Once a valid file name has been entered, read each character one at a time from the file and process as seen below.

```
fstream inFile(fileName, ios::in);  ///< fstream for file
// if the file could not be opened, return false
if(inFile.fail())  return false;

char ch;   ///< character from the file

// read characters until the end of file or list is full
ch = inFile.get();
while(!inFile.eof() && alphaNum < MAX_LIST_SIZE)
{
   if(isalpha(ch))
      addCharToList(tolower(ch), alphaList, alphaNum);
   // get the next character from the file
   ch = inFile.get();
}
inFile.close();
return true;
```

There are three additional functions referenced in the function above. The function `isalpha(ch)` will return `true` if the character `ch` is a letter, `false` otherwise. The function `tolower` will return the lowercase version of a letter. Both functions `isalpha(ch)` and `tolower(ch)` are already provided for you (thanks `glibc`). The function `addCharToList` is a function you must write, but the code below should give you are start.

```
bool addCharToList(char ch, CharFreq list[], int& num)
{
   int loc = search(ch, list, num);
   if(loc != -1)
   {
      // ch already in list, update the count
   }
   else if(num < MAX_LIST_SIZE)
   {
      // ch not in the list, so add it
      num++;
   }
   else
   {
      cout << "list out of space \n";
      return false;  // some badness happened Danny, return false
   }
   return true;  // process correctly, return true
}
```

The function does reference `search()`, which you must write.

3 Programming Points

You **must** adhere to all of the following points to receive credit for this lab assignment.

1. Create a directory `Lab5` off of your `CSC112` directory to store your program in a file called `lab5.cpp`
2. Your program must be modular in design.
3. Your `main` function can only consist of variable declarations, function calls, and control structures (no input or output in the `main` function).
4. Your program must compile cleanly, no errors or warnings are allowed.
5. Your program must adhere to documentation style and standards. Don't forget function headers and variable declarations.
6. **Turn-in** a print-out of your program source code (`lab5.cpp`). In addition, copy your program source code to your `Gade/Lab5` directory.