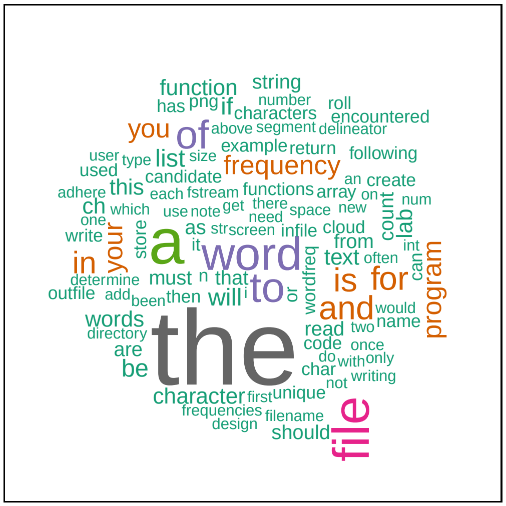
CSC 112 Lab 6
Word Clouds

Spring 2015

Due
3:00pm Friday, 3/20

1 Introduction

A word cloud is a visual representation of the text contained in a file, where the importance of each word is shown with font size or color. For example, the frequency of the word (how often it occurs in the file) can be associated with a font size and color, where larger fonts would represent words that occurs more often. The word cloud for the text of this assignment is given below. For this lab you will write a program to determine the word frequency of a text file, then use another program to create the word cloud (saved as a png file).

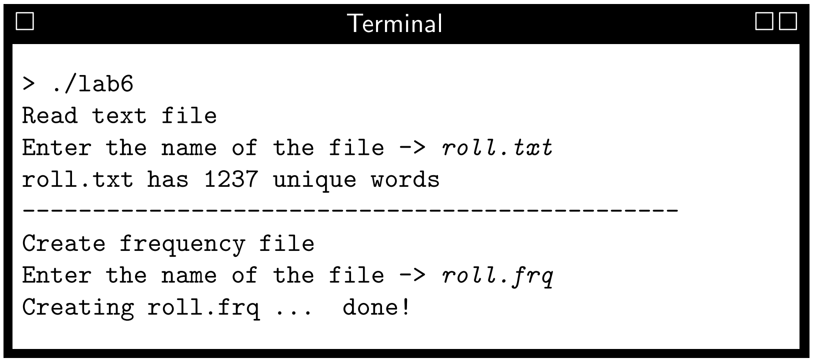


Word cloud for the text in this document.

1.1 Word Frequency

We are interested in determining the word frequency (a count of the number of times a word is used) of a text file. The program will first prompt the user for the name of a text file. If the file does not exist, re-prompt until a legitimate file name is provided. Afterwards, read the file and process every word in the file. Once the the frequency has been determined, print the number of words found to the screen. Then prompt the user for an output file name and write the word frequencies to the file. The words must be sorted in descending order based on the frequencies. For example, assume the user enters `roll.txt` for the text file name and wants to create `roll.frq` as the frequency file. The following would be the result.

screen output



```
> ./lab6
Read text file
Enter the name of the file -> roll.txt
roll.txt has 1237 unique words
----------------------------------------------
Create frequency file
Enter the name of the file -> roll.frq
Creating roll.frq ...  done!
```

`roll.frq`

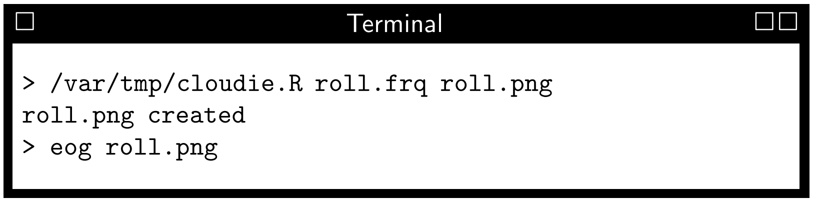
```
the 215
a 98
up 92
to 85
of 80
april 78
retrieved 77
jump 66
astley 59
on 55
⋮
```

The word frequency file should contain one line per unique word found, where each line has the word, a space, then the count. The first 10 lines of the `roll.frq` file are shown above.

1.2 Word Frequency

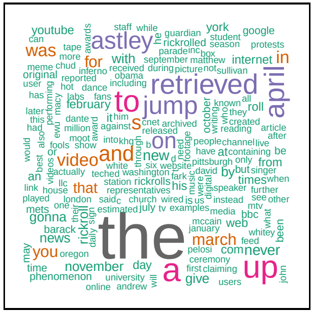
The word cloud can be created using the frequency file described in the previous subsection. The program `cloudy.R` will take a word frequency file and generate a png (graphics format) file. The `eog` command will display the image to the screen.

screen output



```
> /var/tmp/cloudie.R roll.frq roll.png
roll.png created
> eog roll.png
```

`roll.png`



2 Program Design

Although word frequency is a simple task to describe, dealing with C-strings (as you found-out on the test) can be problematic. 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 word appears in the file. The word list should store two items per element, the word (C-string) and the count. Use the following `struct` to store the two items.

```cpp
#include <iostream>
#include <fstream>
// maximum string size
#define MAX_STRING_SIZE 20
// maximum word list size
#define MAX_LIST_SIZE 5000
using namespace std ;

//=== data type to store a word and its count =================
struct WordFreq
{
    // constructor for the WordFreq type, just a good idea...
    WordFreq(){  word[0] = '\0';  count = 0;  }

    // data members
    char word[MAX_STRING_SIZE];      ///< the word
    int  count;                      ///< the word frequency
};
```

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. Note, the `struct` includes a constructor that initializes each instance of the type (a good idea for C-strings). You should then create an array for the unique words in the `main` as follows.

```cpp
WordFreq list[MAX_LIST_SIZE];  // list of unique words
int num = 0;                   // number of unique words
```

2.2 Reading the File and Processing Words

Reading the file and processing words 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. Once a valid file name has been entered, read each character one at a time from the file and build strings.

Consider the characters in the file as a long array that your program will process one element at a time.

text file

```
this is, a
gr8 lab.  OK?
```

array representation

’t’	’h’	’i’	’s’	’ ’	’i’	’s’	’,’	’ ’	’a’	’\n’	’g’	’r’	’8’	’ ’	’l’	’a’	’b’	’.’	’ ’	’ ’	’O’	’K’	’?’	’\n’

Scanning the array from left to right (which is how characters are read from the file) a candidate word has been read once a word delineator is encountered. For example, a space `' '` is a word delineator; therefore, the string `"this"` in the array above is a candidate word. Other delineators include commas, periods, new lines, etc... (you need to develop a complete list). Once a candidate word is encountered, then determine if is it a word. For this lab assume any string that consists only of letters is a word. For example `"gr8"` is not a word, but `"OK"` would be a word for this lab. The following code segment will read characters from a file and produce candidate words. Note, you need to add to the set of word delineators.

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

char str[MAX_STRING_SIZE]; ///< current string
int n;                     ///< length of string
char ch;                   ///< current character

// initialize string
str[0] = '\0';
// get a character from the file
ch = inFile.get();
while(!inFile.eof())
{
   // if ch is word delineator (you should add to the OR statement)
   // then you have a candidate word
   if(ch == ' ' || ch == '\n')
   {
      // if string is a word, add to list
      if(isWord(str))
         addWordToList(str, list, num);

      // reset the string
      str[0] = '\0';
   }
   else
   {
      // add new character to end of the string, if there is space
      n = strlen(str);
      if(n < MAX_STRING_SIZE - 1)
      {
         str[n] = tolower(ch);
         str[n + 1] = '\0';
      }
   }
   ch = inFile.get();
}
inFile.close();
return true;
```

There are two additional functions referenced in the code segment above that you must write.

- `isWord(char str[])` returns `true` if the C-string `str` is a word, otherwise it returns `false`
- `addWordToList(char str[], WordFreq list[], int& num)` updates the word list `list` with the C-string `str`, which is similar to `addCharToList` from lab 5.

2.3 Other Functions and Writing to a File

Similar to the previous lab, there are several other functions you will need to write for this lab assignment. You will write a function to sort the word list based on the frequencies and a function for writing the word list to a file. Note, writing to a file is the same as writing to the screen. The following is a code segment that opens a file and writes the word list to the file.

```
char fileName[MAX_STRING_SIZE];
fstream outFile;

do
{
   readFileName(fileName);
   outFile.open(fileName, ios::out);
}
while(outFile.fail());

for(int i = 0; i < num; i++)
   outFile << list[i].word << ' ' << list[i].count << '\n';

outFile.close();
```

3 Programming Points

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

1. Create a directory `Lab6` off of your `CSC112` directory to store your program in a file called `lab6.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** (copy to your `Grade/Lab6` directory) a word cloud png (image file) of the `wakebaseball.twt` text file, which is available from the course web-site.

7. **Turn-in** a print-out of your program source code (`lab6.cpp`). In addition, copy your program source code to your `Grade/Lab6` directory.