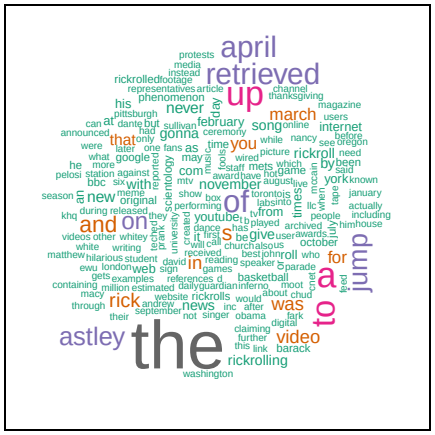
CSC 112 Lab 7
Stop it Vanra!
Spring 2015

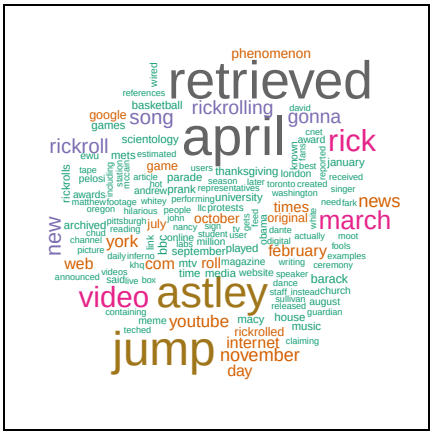
Due
3:00pm Friday, 3/27

1 Introduction

As described in the previous lab, 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. In lab 6, the frequency of the word was used to determine the word size and color. Typically certain or uninteresting or common words are filtered out to provide a better representation of the document. These words are referred to as *stop words* and include "the", "is", "that", etc... For this lab you will generate word clouds with stop words removed.



roll.txt



roll.txt without stop words.

1.1 Word Frequency without Stop Words

Similar to lab 6, we are interested in determining the word frequency (a count of the number of times a word is used) of a text file with the stop words removed. The program will accept three command line arguments; the text file name, the stop words file name, and the resulting word frequency file name. If the user does not provide the three arguments, then the program should stop (do **not** re-prompt) and display how to properly execute the program (explaining the command line arguments and the order). Similarly, if any of the files cannot be opened, stop the program and explain the error. If the arguments are correct, the program should read the text file (first file argument) and process every word. Once the the frequency has been determined, print the number of words found to the screen (count **includes** stop words). Afterwards process the stop words file (second file argument), remove all the stop words from your list, and redisplay the word count. Finally, write the final list to the frequency file (last argument file) and indicate this on the excreen. For example, assume the user wishes to process `roll.txt` as the text file, `stop.txt` contains the stop words, and `foll.frq` is the resulting file. The following would be the result.

screen output

```
Terminal
> ./lab7 roll.txt stop.txt roll.frq
roll.txt has 1237 unique words
-----------------------------------------------
without stop words (read from stop.txt)
roll.txt has 1079 unique words
-----------------------------------------------
Creating roll.frq ...  done!
```

`roll.frq`

```
april 78
retrieved 77
jump 66
astley 59
rick 39
video 38
march 31
new 24
rickroll 23
song 22
⋮
```

2 Program Design

Managing the word frequency list will be very similar to the previous lab except **the list must be dynamically allocated** such that there is no wasted space (logical and physical size are always equal). In addition, your program **must** adhere to the following program design requirements.

2.1 A Dynamic Array for Word Counts

As characters are read from the file, you will keep track of the number of times a word appears. This word list should store two items per element, the word (C-string) and the count. You can use the `struct WordFreq` from lab 6 to store the two items. Note, the C-string for the word is a static `char` (physical size is `MAX_STRING_SIZE`) which is acceptable for this assignment; however, the list of `WordFreq` **must** be dynamic. As a result, the `WordFreq`list will be declared as a pointer in the `main` function, as seen below.

```
1 WordFreq* list = 0;  ///< dynamic list of unique words
2 int num = 0;         ///< number of unique words
```

2.2 Multiple Files and `makefile`

The source code for this assignment must be appropriately divided into the following 3 files.

- `main.cpp` contains the main function.
- `words.h` contains the word function prototypes (declarations).
- `words.cpp` contains the word function definitions.

3 Programming Points

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

1. Create a directory `Lab7` off of your `CSC112` directory to store your program files
2. The assignment will consist of 3 files.
   - `main.cpp` contains the main function.
   - `words.h` contains the word function prototypes (declarations).
   - `words.cpp` contains the word function definitions.
3. Your program must be modular in design.
4. Your `main` function can only consist of variable declarations, function calls, and control structures (no input or output in the `main` function).
5. Your program must compile cleanly, no errors or warnings are allowed.
6. Your program must adhere to documentation style and standards. Don't forget function headers and variable declarations.
7. **Turn-in** (copy to your `Grade/Lab7` directory) a word cloud png (image file) of the `wakebaseball.twt` text file with the stop words removed, which is available from the course web-site.
8. **Turn-in** a print-out of your program source code (`main.cpp`, `words.cpp`, `words.h`, and `makefile`). In addition, copy your program source code to your `Grade/Lab7` directory.