Project Report by Alex D. Santos and Iván Jiménez
December 11, 2012

I. Introduction:

The purpose of this project was to create a search engine. This project involved creating a storage structure, a user interface and a search methodology. First, the engine starts building the storage structure. Then, it displays the user interface that allows the user to look up to two words to see if there are files associated with them. Finally, it displays the qualifying files and grants the user the freedom to keep searching or to quit.

II. Contributions by each team member

Iván Jiménez worked on the user interface and cleaning part. Iván created the logo and verified that the user interface was complete, neat and correct. Also, he tested the program to see if it had errors, making modifications to improve the comfort while using it. Iván also worked on the function that creates the structure that stores the stop words and was in charge of orthography and program documentation.

Alex D. Santos worked on the storage structure, optimization, and code encapsulation. He made the functions that read the files, the words in the files, the function that cleaned the words before storing them, the function that receives the inputs of the user and validates the input. Also, he made the storage structure, the functions that searched for the qualifying files, the function that displays them and encapsulated all the code into functions.

III. Summary of Design:

A. Client Program

The client program is called client.cpp. This program creates a storage structure that relates all the existing words that are valid (not stop words) to all the files in which they appear and how many times they appear in said files. Furthermore, it calls all the necessary functions to allow the search to start (this function is called “setEngine”). When the engine is done building, it displays how much time it took to build and the logo of the search engine “B-Bam! S. Engine” (this function is called “displayLogo”). Afterwards, it explains to the user how to enter the words and waits for the user to enter the input (this function is called “getWords”). After getting the inputs and dividing it into words, the engine checks if there are one or two words. If there is one or two words, it searches for all of the qualifying files and stores them in a map. If there are more than two words, the engine asks for the inputs again (this function is called “getResults”).

B. Process method:

setEngine():

- Calls all the necessary functions to set the engine and allow the search
- Stop(): Post-condition:
  - Eliminates every character of the word that is not a letter
  - Converts every character into lower case.
- Getdir():Receives the name of a directory as a parameter and returns (via a second parameter) a vector of strings that contains the names for each archive in the directory.

getWords():

- Receives an empty vector of strings.
- Inputs: stores the inputs provided by the user.
- Breaks down the input into separate words.

getResults():

- Receives word one, word two, files_freq and results.
  - Word one and two: provided by the user input
  - files_freq: contains the words, files, frequency
    - word => files => frequency
  - results: contains the qualifying files
- Searches for all the files related to the two words. Then looks for the files that have the two words and stores them in a map.

displayResults():

- Receives results.
  - results: contains the qualifying files
- Displays the three files with the highest frequency.

C. Parts of the program and pseudo-code:

1. Build:

a. Starts the timer.
b. First, it reads all the files from the directory.
c. While it is reading all the files, all the words in the files are read.
d. While reading the words, all the characters that are not letters are eliminated.
e. Then, it checks if the words are not stop words.
f. If the words are not stop words, they are added to an unordered map.
g. Later, it adds the name of the file in which it was found and increments the counter.

h. The engine will continue adding and counting until it finishes reading all the words of all the files.
i. The timer is stopped.
j. Finally, it displays the time it took to build.

2. Search:

a. Displays the logo.
b. It asks the user to enter one or two words, separated by a space.
c. It separates the input into separate words.
d. If there is one word, it searches for the intersection of the word with itself.
e. If there are two words, it searches for the intersection of the words.
f. It stores the intersection in a multimap.

3. Display:

a. Displays the three files with the highest frequency using a reverse iterator.
b. It asks the user if he wants to perform another search.
c. If the input is "yes", it runs the program starting from the search.
d. If the input is "no", it terminates.

During the creation of the search engine, several versions of the code (which makes up the project) were produced. The following table shows the description of each version of code, the error they contain and the steps implemented to fix these particular errors:

Version	Error	Debugging
1	There was an error. Originally, the program displayed all the words and their frequency.	We changed the code to be able to display the correct correlation between words, files and frequency.
2	The program took several minutes to process and display every word.	We studied the functions in the code in an effort to afford space and time. We redesigned part of the code: eliminating loops and vectors, modifying several functions given by the professor.

3	There was an error. The program processed every word, without taking into account the stop words.	We used a storage structure (unordered_set), so that it would it would contain the stop words. We took this structure as a guide to eliminate all unnecessary words in the processing phase.
4	There was an error. The command "cin.getline()" did not function as intended.	We added a counter and an "if" statement to the function implementing the faulty command line.
4.1	There was an error. In sorting the results, the map that was used eliminated files due to repeated frequencies.	We utilized a different data structure (multimap).
4.2	There was an error. The program took into account the lowest frequency found when displaying the three files.	We changed the normal iterator to a reversed iterator.

The search engine developed in this project required different tests to ensure that the correct result was displayed upon the user's request. The following table shows several tests performed at different stages of the development process:

Testing	Success/Failure
We tested to see if the maps used in the early versions of our search engine contained the correct information.	The testing was successful because we received the desired output:

We tested the program using 1 word.	The testing was successful in processing the word and providing the correlation from word to file to frequency.
We tested the program using 2 words.	The testing was successful in processing the words and displaying the sum of the frequencies of both words.
We tested the program by trying to accomplish more than one search in a single execution.	The testing was unsuccessful due to the "cin.getline()" problem that was previously mentioned, which did not allow a follow-up user input after a search was executed. In the later versions of the program, this test was successful.

Input	Results (if successful)
"pepe"	Returned 23 results. Files with highest frequency: PEpE-le-Moko.txt Stin-pagida-tou-sex-kai-tou-eglimators.txt Walls-Have-Ears.txt
"Pepe"	Returned 23 results. Files with highest frequency: PEpE-le-Moko.txt Stin-pagida-tou-sex-kai-tou-eglimators.txt Walls-Have-Ears.txt
"pepe", "moko"	Returned 1 result. File with highest frequency: PEpE-le-Moko.txt
"PEPE", "MOKO"	Returned 1 result. File with highest frequency: PEpE-le-Moko.txt

"pepe", "hola", "moko"	"Enter one or two words seperated by a space:" (input validation)
"asdfasdfasdfasdfasdf"	"Your search returned 0 results. Showing you all the results: Want to keep searching? Enter 'y' if yes or 'n' if no"
"asdfasdf", "asdfasdfasdf"	"Your search returned 0 results. Showing you all the results: Want to keep searching? Enter 'y' if yes or 'n' if no"

IV. User Manual

A. Instructions

1. **Open the terminal**
2. **Change the directory to the directory containing the files**
3. **Enter "make"**
4. **Enter "./client"**
5. **Wait for the engine to build**
6. **Enjoy :)**