Adam Camerer
Mukund Telukunta
CS1585
07/23/2022

Project 2 Results

Project B: C++ Standard Library

Approach

Using cstdlib, I relied primarily on map for storage, cctype for filtering words, fstream for reading files, and string for storing the 'key' of the string, and iostream to output the results on the console.

For the project, I used a map to store occurrences of unique words, and then outputted them using an iterator. Since iterators are more generalizable, and I wanted to apply the knowledge I had learned, I used them to go through the map and output the results.

To not have to scroll as much, I outputted 3 key-value pairs on each line, separated by two tabs.

```
//instantiate iterator for map
template<typename K, typename V>
using MapIterator = typename std::map<K,V>::const_iterator;


//Print map of words (when finished)
template<typename K, typename V>
void PrintMap(const std::map<K,V>& m)
{
  int i = 0;
    for (MapIterator<K, V> iter = m.begin(); iter != m.end(); iter++)
    {
        std::cout << "Key: " << iter->first << " "<< "Values: " << iter->second;
        if(i%3==2)  std::cout<<std::endl;
        else  std::cout<<"\t\t";
        i++;
    }
```

To process the words, I first created a fileStream object for the input file, and then iterated through each 'string' in the file (characters non separated by whitespace), which made it possible to select individual words

```
int main(void)
{
    //file
    static const char* fileName = "4507.txt.utf-8";

    // Will store the word and count.
    map<string, unsigned int> wordsCount;


    {
        // Begin reading from file:
        ifstream fileStream(fileName);

        // Check if file opened
        if (fileStream.is_open())
            while (fileStream.good())
            {
                // Store the next word in the file in a local variable.
                string word;
                fileStream >> word;
```

Once a ‘word’ was selected, I would begin processing it to remove ‘basic’ punctuation, i.e. not double hyphens, since that required a more specialized technique. I ignored web addresses (https), since they are usually unique (web addresses aren’t even real words). It would be more difficult to add the : and / characters than ignore them altogether.

Using ispunct from cctype, I got a larger pool of characters to remove. Then, I whittled down the pool of characters, selecting all at the beginning or end of the word, or non hyphens between the beginning and end. Then, I removed the characters, and continued on.

After this, I then relooped over the loop again, converting all uppercase characters to lowercase. If I put the punctuation removal and lowercase in one loop, it would be more confusing.

Removing double hyphens was more complex. Since ‘basic’ punctuation was already removed, it made the job easier. To remove them, two words were created, so each would need to be added to the tally separately, and not as one big glob. To do this, I selected only words with double hyphens, and then took the substring before the double hyphen, added it to the map, and then only kept the part after the double hyphen.

```
74                  if( !(word.find(webAddress) != std::string::npos) )
75                  for(int i = 0; i<len; i++)
76                  {
77                    //delete punctuation (except in special cases)
78
79                    if(    ispunct(word[i]) && ( (i==0 || i == len-1) || (word[i] != '-') )    )
80                    {
81                      word.erase(i,1);
82                      len = word.size();
83                      i--;
84                    }
85                  }
86
87                  //convert uppercase letters to lowercase
88                  for(int i = 0; i<len; i++)
89                  {
90                    word[i] = tolower(word[i]);
91                  }
92
93
94                  //Deal with double hyphen
95                  for(int i = 0; i<len; i++)
96                  {
97                    if(    (word[i]=='-' && word[i+1]=='-')    )
98                    {
99
00                      string lower = word.substr(0,i);
01                      addToMap(wordsCount, lower);
02                      word.erase(0, i+2);
03                      len -= i+2;
04                      i=0;
05                    }
06
```

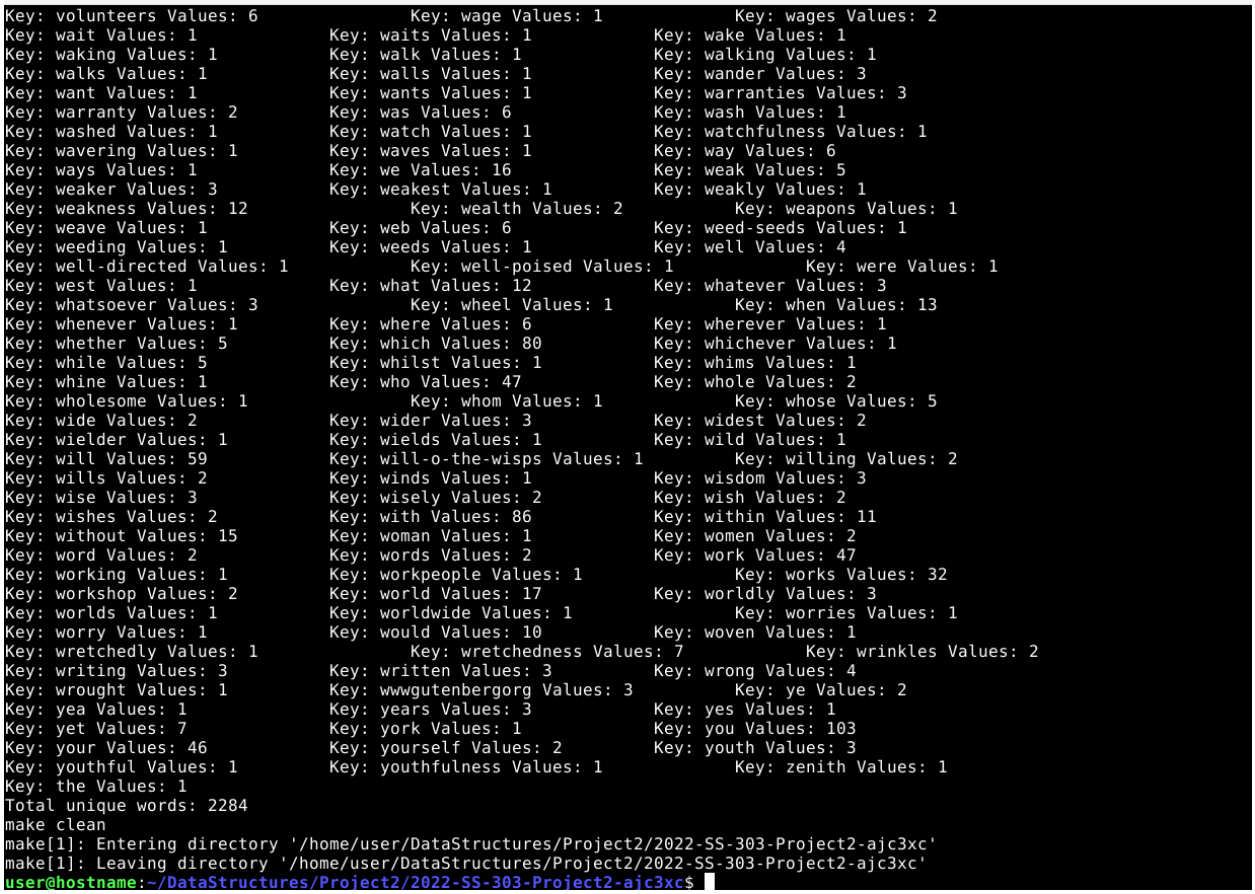
After this, I added the filtered word to the map. Once all the words were filtered and mapped, I printed the map. If the file wouldn't open, an error message would be printed. After this, the program ended.

```
                //add (remaining) word to map
                addToMap(wordsCount, word);
            }
        else  //File not openable
        {
            cerr << "Couldn't open the file." << endl;
            return EXIT_FAILURE;
        }

        // Print the words map.
        PrintMap(wordsCount);
    }

    return EXIT_SUCCESS;
}
```

Results



Compiler (Makefile)

```
.PHONY: clean

clean:
  -@rm -f wordMap
  -@rm -f *.o
  -@rm -f *.aux
  -@rm -f *.log
  -@rm -f *.dvi

program:
  g++ wordMap.cpp -o wordMap
  ./wordMap
  make clean

document:
  pdflatex document.tex
```