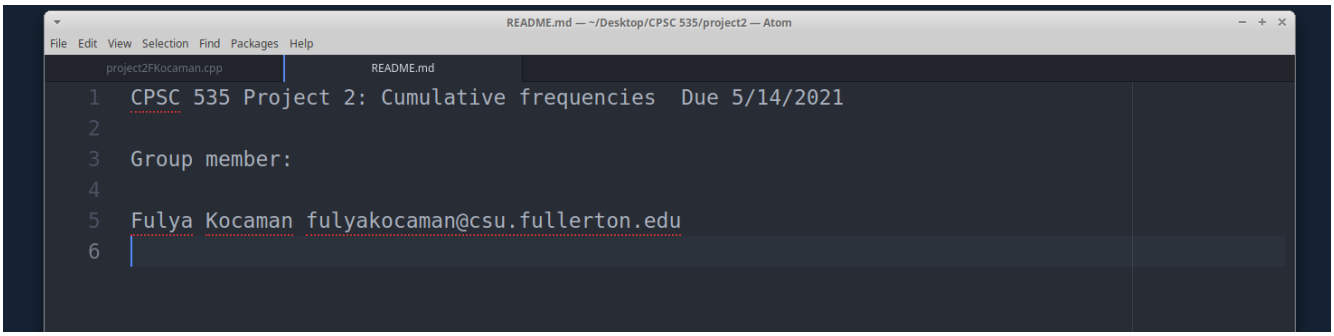
In this project I designed and implemented Cumulative frequencies of similar words algorithm related to strings using C++. This algorithm uses two lists one of words and their frequencies and the other of pairs of similar words and prints a new list of cumulative frequency of each set of similar words. In the output list, the words that are the earliest in the alphabet will represent of its set in a sorted order.



1. Pseudocode:

Input: Two lists: Words_Frequencies [] pairs with size m > 0 and Synonyms [] pairs with size n > 0.

Output: the new list of Cumulative frequencies [] pairs with size k > 0

```
// Function returns the number of words in a file
def numWords(S):
  num = 1
  string sentenceRead
  open file
    while !file.eof():
      getline(file, sentenceRead)
      ++num
    endwhile
  close file
 return num
enddef


// Function returns pairs of words from the word frequency file

def vector<pair<string, int>> getWordFreqPairs(const string &fileName, int wordSize):

  // Stores each sentence read from a file
  string sentenceRead
```

```
  /*(key, value) pairs. The key is a string and the value is its corresponding
  frequency */
  vector<pair<string, int>> vecPairs;
  string key[wordSize];
  int value[wordSize]; // corresponding frequency of the string
  open file
   for i = 0 to wordSize – 1 do:
       getline(file1, sentenceRead)
       key[i] = sentenceRead
       getline(file1, sentenceRead)
       value[i] = stoi(sentenceRead)
       vecPairs.push_back(make_pair(key[i], value[i]))
    endfor
  close file
 return vecPair
enddef

// Function returns pairs of words from the synonyms file
def vector<pair<string, string>> getStringPairs(const string &fileName, int wordSize):
  // Stores each sentence read from a file
  string sentenceRead;
  /*(key, value) pairs. The key is a string and the value is its synonym */
  vector<pair<string, string>> vecPairs;
  string key[wordSize];
  string value[wordSize]; // corresponding synonym string
  open file
    for i = 0 to wordSize – 1 do:
      getline(file1, sentenceRead)
      key[i] = sentenceRead
      getline(file1, sentenceRead)
      value[i] = stoi(sentenceRead)
      vecPairs.push_back(make_pair(key[i], value[i]))
    endfor
  close file
 return vecPair
enddef

// Function prints original pairs of words with their frequencies
def printFreqPairs(vector<pair<string, int>> wordFreq):
    for &it : wordFreq do:
        print( "[name: " << it.first << "] = " << it.second )
    endfor
print( "\nsize of word frequency pairs = " << wordFreq.size())
```

```
// Function prints original pairs of words with their synonyms
def printSynonymsPairs(vector<pair<string, string>> synonyms)
   for &it : synonyms do:
      print( "[name: " << it.first << "] = " << it.second )
   endfor
print( "\nsize of word frequency pairs = " << wordFreq.size())

def main(int argc, char **argv):
  if (argc < 3): // Sanity check -- make sure the user provided all of the required arguments
   fprintf(stderr,
         "USAGE: %s <StringS FILE NAME> <StringPairsLS FILE NAME> \n", argv[0])
   exit(1)
  endif
// Stores the file name of the input pairs of words and their frequencies
  string wordFreqFileName = argv[1]
  // Stores the file name of the input pairs of synonyms
  string synonymsFileName = argv[2]
  // The size of the vector of words
  int size1 = numWords(wordFreqFileName) / 2
  // The size of the vector of words
  int size2 = numWords(synonymsFileName) / 2
  // Vector of strings read from the stringS file
  vector<pair<string, int>> wordFreqVec = getWordFreqPairs(wordFreqFileName, size1)
  // Vector of strings read from the stringS file
  vector<pair<string, string>> synonymsVec = getStringPairs(synonymsFileName, size2)
  // prints the original word frequency pairs
  printFreqPairs(wordFreqVec)
  // prints the original word synonyms pairs
  printSynonymsPairs(synonymsVec)

// creates unique word frequency pairs
  for k = 0 to wordFreqVec.size() – 1 do:
   for i = k + 1 to wordFreqVec.size() do:
     if ((wordFreqVec[k].first == wordFreqVec[i].first) &&
        (wordFreqVec[k].second == wordFreqVec[i].second)) then
        wordFreqVec.erase(wordFreqVec.begin() + i)
      endif
    endfor
   endfor

  // creates unique word synonym pairs
  for k = 0 to synonymsVec.size() – 1 do:
```

```
  for i = k + 1 to synonymsVec.size() do:
    if ((synonymsVec[k].first == synonymsVec[i].first) &&
        (synonymsVec[k].second == synonymsVec[i].second)) then
       synonymsVec.erase(synonymsVec.begin() + i)
     endif
   endfor
 endfor

 // creates an adjacency matrix
 vector<vector<string>> adjMatrix
 adjMatrix.resize(wordFreqVec.size())

// populates the adjacency matrix with rows from wordFreqVec
 for (int k = 0 to wordFreqVec.size() do:
  adjMatrix[k].push_back(wordFreqVec[k].first)
  // Now, in the adj matrix each row populated with wordFreq names
 endor

// adds edges to wordFreqVec from synonym pairs
 for j = 0 to synonymsVec.size() do:
  // outer loop index j for the each synonym from synonymsVec
  for i = 0 to wordFreqVec.size() do:
    // inner loop index i for the rows of the adj matrix
    if (adjMatrix[i][0] == synonymsVec[j].first) then
      // add an edge to the matching synonym word
      for k = 0 to adjMatrix.size() do:
        // find the word corresponding to the to the matching synonym word
        // to add an edge symmetrically, so index k keeps track of the
        // corresponding synonym word
        if (adjMatrix[k][0] == synonymsVec[j].second) then
          int index = adjMatrix[k].size()
          for m = 0 to index do:
            // adds an adge to the matching synonym word
            adjMatrix[k].push_back(adjMatrix[i][m])
            // adds an adge to the matching  word
            adjMatrix[i].push_back(adjMatrix[k][m])
          endfor
          break
        endif
     endfor
      break;
    endif
  endfor
 endfor
```

```
  // removes repeating words in the adjacency matrix if any
  for m = 0 to adjMatrix.size() do:
    // outer loop index m for the row of adjMatrix
    for k = 0 to adjMatrix[m].size() do:
      // index k keeps track of the first word of each row m
      for i = k + 1 to adjMatrix[m].size() do:
        // index i keeps track of the next word of each row m to compare if they
        // are the same
        if (adjMatrix[m][k] == adjMatrix[m][i]) then
          // if same, remove the next word from that row
          adjMatrix[m].erase(adjMatrix[m].begin() + i)
        endif
      endfor
    endfor
  endfor

// sorts the words in each row of the adjMatrix
  for i = 0 to adjMatrix.size() do:
    sort(adjMatrix[i].begin(), adjMatrix[i].end())
  endfor

  // sorts the rows of the adjMatrix
  sort(adjMatrix.begin(), adjMatrix.end())

  // removes the same rows if there is any
  adjMatrix.erase(unique(adjMatrix.begin(), adjMatrix.end()), adjMatrix.end())

// removes the rows that are subsets of the other rows which have at least
  // one common word
  for k = 0 to adjMatrix.size() – 1 do:
    // outer loop index k for the row of adjMatrix
    for m = 0 to adjMatrix[k].size() do:
      // index m to keep track for each row m
      for i = k + 1 to adjMatrix.size() do:
        // index i for the next row after row k
        for j = 0 to adjMatrix[i].size() do:
          // index j to keep track for each row j
          if ((adjMatrix[k][m] == adjMatrix[i][j]) && (adjMatrix[k].size() > adjMatrix[i].size())) then
            // if they have a common word and size of row k > size of row i, removes row i
            adjMatrix.erase(adjMatrix.begin() + i)
            break
          endif
```

```
      if ((adjMatrix[k][m] == adjMatrix[i][j]) && (adjMatrix[k].size() < adjMatrix[i].size())) then
        // if they have a common word and size of row k < size of row i, removes row k
        adjMatrix.erase(adjMatrix.begin() + k)
        break
      endif
    endfor
  endfor
 endfor
endfor

print( "The cumulative frequencies: \n")
// keep track of each cumulative frequency of similar groups of words
count = 0;

for i = 0 to adjMatrix.size() do:
 // outer loop index i for the row of adjMatrix
 for it1 = adjMatrix[i].begin() to it1 != adjMatrix[i].end() do:
  // iterator 1 goes through each row of the adjMatrix
  for &it2 : wordFreqVec do:
   // iterator 2 goes through each row of wordFreqVec
   if (*it1 == it2.first) then
   // if finds a match, adds up to the grand total
    count += it2.second
   endif
  endfor
 endfor
 // print the first word from each sorted similar row along with each group's cumulative frequency in a
        // sorted fashion
 print(adjMatrix[i][0] << " = " << count)
 count = 0
endmain
```

2. How to Run the Code:

C++ language is used in this project. From the Linux terminal:

To compile the greetingsCards.cpp use the command: ++ project2FKocaman.cpp -o project2

To run the program, use the command:
./ project2 <wordFreq FILE NAME> <synonyms FILE NAME>
, where < wordFreq FILE NAME> is the name of the file containing of words and their frequencies and < synonyms FILE NAME> is the name of the file containing pairs of similar words.

3. Snapshots of the Code Executing for the Three Given Examples

Example 1:

Input:

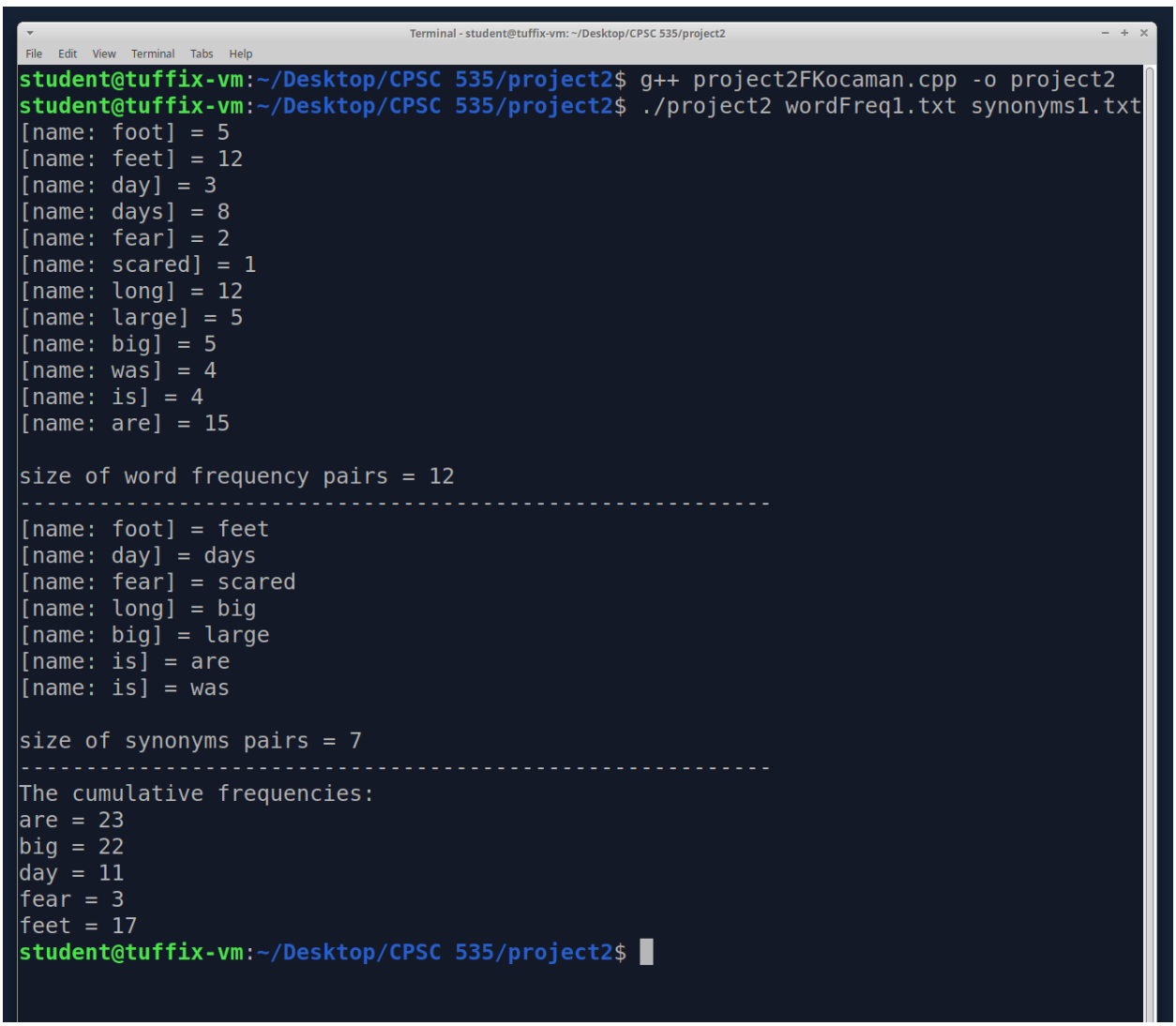
Words_Frequencies: WF[] = { (“foot”, 5), (“feet”, 12), (“day”, 3), (“days”, 8), (“fear”, 2), (“scared”, 1), (“long”, 12), (“large”, 5), (“big”,5), (“was”, 4), (“is”, 4), (“are”, 15)} of size 12

Synonyms: SYN[] = { (“foot”, “feet”), (“day”,“days”), (“fear”, “scared”), (“long” ,“big”), (“big” , “large”), (“is”, “are”), (“is”, “was”) } of size 7

Output: CF[] = { (“are”, 23), (“big”, 22), (“day”, 11), (“fear”,3), (“feet”, 17) } of size 5.

My code lists the representative words of each set in a sorted order.

The Example 1 Output:



Example 2:

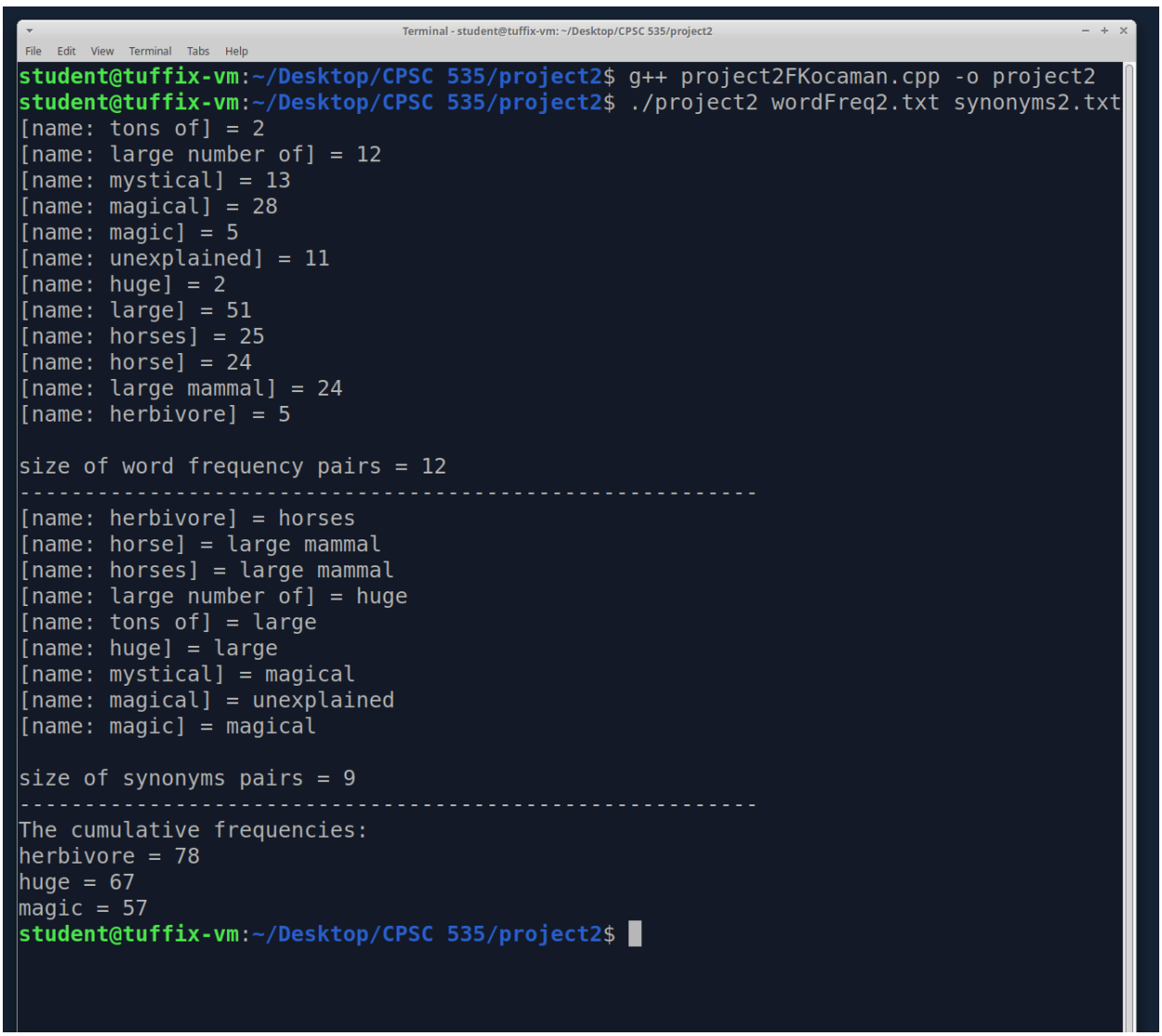
Input:

Words_Frequencies: WF[] = { (“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“horse, 24), (“large mammal”, 24), (“herbivore”, 5)} of size 12

Synonyms: SYN[] = { (“herbivore”, “horses”), (“horse”,“large mammal”), (“horses”, “large mammal”), (“large number of” ,“huge”), (“tons of” , “large”), (“huge”, “large”), (“mystical”, “magical”) , (“magical”,”unexplained”), (“magic”, “magical”)} of size 9

Output: CF[] = { (“herbivore”, 78), (“huge”, 67), (“magic”,57)} of size 3

The Example 2 Output:



Example 3:

Words_Frequencies: WF[] = { (“tons of”, 2), (“large number of ”, 12), (“mystical”, 13), (“magical”, 28), (“magic”, 5), (“unexplained”, 11), (“huge”, 2), (“large”, 51), (“horses”, 25), (“horse, 24), (“large mammal”, 24), (“herbivore”, 5), (“large number of”,12)} of size 13

Synonyms: SYN[] = { (“herbivore”, “horses”), (“horse”,“large mammal”), (“horses”, “large mammal”), (“large number of” ,“huge”), (“tons of” , “large”), (“huge”, “large”), (“mystical”, “magical”) , (“magical”,”unexplained”), (“magic”, “magical”), (“horse”,”large mammal”)} of size 10

Output: CF[] = { (“herbivore”, 78), (“huge”, 67), (“magic”,57)} of size 3

The Example 3 Output: