CS203 Assignment 1: Word Statistic - Huu Khang Nguyen 7402909

1/ Problem:

Write a program that will read from a text file (line by line, word by word or character by character), and generate a statistical output of this text file. The output should contain:

- The top ten words with the highest frequency along with their frequency
- The last ten words with the lowest frequency along with their frequency
- All the unique word (frequency = 1)

To do this, the program might need to:

- Parse the input
- Store the frequency of each word.
- Sort the words based on the decreasing order of the word's frequency. If exist multiple words have the same frequency, sort them alphabetically

Note: The program might discard punctuation characters, and in case these characters exist between 2 words, treat it as a single word.

Constraints:

- The program cannot use the built-in String data structure.
- The program cannot use the built-in STL data structure and any built-in algorithms.
- The text file will contain alphabetical characters, uppercase/lowercase characters, non-alphabetical characters (for e.g: ?!,.'"-,...), and numeric character
- 0 <= n <= 50000 (n as in number of words)

2/ General Direction:

Approach

- Reading, parsing and storing the input in a string pool
    - Read the input from the text file, loop through **each word** and then **each character**
        - For each character, check **if the character** is an **alphabet character**, if it is, convert it into a lowercase character and append it to the string pool
    - Record the **length** of the word, and append it to another array that store the length of each word. This array will be used later to construct the complete word.
- Initialising a hash map, loop through the string pool, and construct each word from the string pool, for each word:
    - Check if the word exists in the hashmap
        - If the word does not exist, create a new entry in the hash map, with initial the count of 1
        - If the word does exist, update the current entry in the hash map by incrementing their count by 1
- Loop through the entries in the hashmap, and construct an array of these entries so that they can be sorted.
- Run the `sort()` function in the array, I will use Merge Sort. during this `sort()` function:
    - Put the element with the highest frequency first, if 2 entries during the comparison have the same frequency:
        - Get the length of the key with a shorter length
        - Loop through every character of the 2 keys, end loop when the current index is the shorter length
            - During this loop, compare the ASCII value (which is an integer type) between 2 characters, 1 from each key.
                - Place the entry that has the key with a smaller ASCII value first before the entry with a larger ASCII value (alphabetically).
- Loop through the sorted entries, starting from index 0 to index 9, and output each element along with their count.
  →Satisfy the requirement of outputting the first 10 elements in the sorted array

- Loop through the sorted entries, starting at the last 9th element of the array (0-indexed), and end at the last element of the sorted array (`entries[array size - 1]`), and output each element along with their count

  → Satisfy the requirement of outputting the last 10 elements in the sorted array

- Loop through the entire sorted entries, starting at the last element of the sorted array (`entries[array size - 1]`). Break the loop when the current element has a count that is not equal to 1.

  → Satisfy the requirement of outputting the last unique element (`count == 1`)

Complexity

Space Complexity:

- Using String Pool to store the input: `O(n)`
- Using HashMap to store the occurrence of every word: `O(n)`
- Space from the Merge sort `O(n)`
- An array to store the sorted hashmap entries `O(n)`

⇒ **Space Complexity**: `O(n) + O(n) + O(n) + O(n)` = `O(4n)` = `O(n)`

Time Complexity:

- Creating a string pool, as we loop and check through every word, and every character: `O(n)`
- Store the occurrence of words with Hash Map:`O(n)`
- Loop through the HashMap and created an Entries array: `O(n)`
- Using Merge Sort to sort the Hash Map: `O(nlog(n))`
  - As I use the hash map to store the word frequency, access the element would be constant: `O(1)`
- Getting the 10 first entries from the sorted array will be `O(n)` (The worst case would be to get all of the words, for example, if there are 10 unique elements from the input, hence O(n))
- Getting the last 10 entries from the sorted array will be `O(n)` [same reason with above]
- Getting all the unique key value entry (`count == 1`) will be `O(n)`

⇒ **Time Complexity:** `O(n) + O(n) + O(n) +  O(nlog(n)) + O(n) + O(n) + O(n)` = `O(nlog(n)) + O(6n)` = `O(nlog(n))`

3/ Pseudo-code:

```
# Main File
import HashMap
import mergeSort

main():
  textFile = "./path-to-txt-file";
  stringPool = new char[100000];
  wordsLength = new char[50000];

  numberOfWords = 0;
  curStringPoolIndex = 0;
  # Build stringpool
  for(word in textFile):
    textLength = 0;
    for(character in word):
      if character is alphabetically:
        stringPool[curStringPoolIndex++] = lowercase(character)
        textLength++;
    wordsLength[numberOfWords++] = textLength

  # Create a Map to store the occurence of words
  hashMap = HashMap()
  curStart = 0
  for(i in range 0 to numberOfWords):
    curString = new char[wordsLength[i]];
    for(j in range 0 to wordsLength[i]):
      curString[j] = stringPool[curStart + j]
    start += wordsLength[i]
    if(hashMap.has(curString)):
      count = hashMap.get(curString)
      hashMap.put(curString, count)
    else :
      hashMap.put(curString, 1)

  # Create an array of map items
  entries = new KeyValueEntry[hashMap.getSize()];
  i = 0;
  hashMapHead = hashMap.head;
  while(hashMapHead is present):
    entries[i++] = hashmapHead
    hashMapHead = hashMapHead.nextIteration
  # sort
  merge_sort(entries, 0, entries.size -1 );

  k = 10
  if hashMap.getSize() < 10
    k = hashMap.getSize()
  for(i in range [0,k)):
    print {entries[i].key} : {entries[i].value}

  for(i in range [entries.size - k to entries.size)):## loop the entries reversely
    print {entries[i].key} : {entries[i].value}

  for(i in range [entries.size - 1 to 0]):
    if entries[i].value not equal to 1:
      break
    else
      print entries[i].key
```

```
# Hash Map File
class HashMap:
  private:
    CAPACITY = 5000;
    Bucket map = new Bucket[5000];
    size = 0

    getHash(word)
      # djb2 algorithm for hashing
      hash = 5381;
      for(character in word):
        hash = hash * 33 + character
      return hash % CAPACITY

  public:
    head = NULL
    end = NULL

    HashMap()
    getEntry(key):
      hash = getHash(key)
      return map[hash].getEntry(key)

    getSize():
      return size

    put(key,value):
      if(has(key)):
        getEntry(key).setValue(value)
      else:
        hash = getHash(key)
        map[hash].addEntry(key,value)
        if(head is equal to NULL):
          head = map[hash]
          end = map[hash]
        else:
          end.nextIteration = map[hash]
          end = end.nextIteration

    get(key):
      if has(key):
        return getEntry(key).getValue()
      return -1

    has(key):
      return getEntry(key) not NULL

class Bucket:
  private:
    begin = NULL
    end = NULL
  public:
    Bucket()
    addEntry(key,value):
      keyVal = new KeyValueEntry(key,value)
      if(begin == NULL)
          begin = keyVal;
          end = keyVal;
      } else {
          end.next = keyVal;
          end = end.next;
      }
    getEntry(key):
      keyVal = begin;
      while(keyVal not NULL){
          if(keyVal is equal key)){
              return keyVal;
          }
          keyVal = keyVal.next;
      }
      return NULL;

class KeyValueEntry
  private:
    key = ""
    value = 0
  public:
    nextIteration = NULL;
    next = NULL;
    KeyValueEntry(_key,_value):
      key = key
      value = value

    setEntry(_key,_value):
      key = _key
      value = _value

    setValue(_value):
      value = _value

    getValue():
      return value

    getKey():
      return key
```

```
# Sort File
import compare from String
merge(arr,start,mid,end):
  leftSubArray = arr[copy from start to mid]
  rightSubArray = arr[copy from mid to end]
  i=0
  j=0
  k=start
  while(i < leftSubArray.size && j < rightSubArray.size):
    if(leftSubArray[i].value > rightSubArray[j].value):
      arr[k] = leftSubArray[i++]
    else if (leftSubArray[i].value < rightSubArray[j].value):
      arr[k] = rightSubArray[j++]
    else
      if(compare(leftSubArray[i].key, rightSubArray[j].key) == -1):
        arr[k] = leftSubArray[i]
      else:
        arr[k] = rightSubArray[i]
  while(i < leftSubArray.size):
    arr[k] = leftSubArray[i]
  while(i < rightSubArray.size):
    arr[k] = rightSubArray[i]

mergeSort(arr,start,end):
  if start >= end return;
  mid = start + (end - start)/2 # this will prevent overflow better than (start + end)/2
  mergeSort(arr,start, mid)
  mergeSort(arr,mid + 1, end)
  merge(arr,start, mid, end)
```

```
# String File
compare(str1, str2):
  shorterLength = MIN(str1.size, str2.size)
  for(i in range (0, shorterLength]):
    if str1[i] < str2[i]:
      return -1;
    else if str1[i] > str2[i];
      return 1;
  return 0 # both strings are the same
```

4/ Data structures used:

Hash Map

- I chose to use hashmap due to its constant lookup time and constant insertion (`O(1)`**). Alternatively**, I can have an array of records storing the word and their value, however, the lookup time of the array would be `O(N)`, which is not as efficient as Hash Map.

- This HashMap will be implemented from scratch, with operations such as `get()` and `put()`
- djb2 hash function will be used to hash the key into the index in the array of the hashmap internally
    - Although rarely, the collision might happen as the index generated from the hash function could potentially already be taken by another entry, no matter how good the hash function is.
        - Solution:
            - Create an array of entries in each bucket of the hashmap. This is the easiest data structure to adopt. However, arrays are fixed size, and this does not make the bucket dynamic. Moreover, there would be some unused space in the array.
            - Create a linked list of entries in each bucket of the hashmap (or each index of the array). When a collision happens, add a new key-value entry at the end of the linked list tail presented at the collision index. **This is my adopted solution** due to it does not create additional unused memory like the array, and its simplicity.
            - As the Linked list lookup time would be `O(n)`, the AVL tree can be used to optimise this even more by `O(log n)`

Array

- I used array to create HashMap and StringPool data structures.

StringPool

- I used an array of char to store the input data, as it is a simple way to store the input. Specifically, I will implement a slightly more optimised version of the String Pool data structure presented in the lecture. The optimisation is more about the space complexity of the data structure, instead of using 3 arrays in totals (char array, int array to store the start of word, and int array to store the length of word), I will use 2 arrays.
    - Initialised an array of characters, and store all the characters tightly together.
    - Initialised a second array of integers that will record the length of each character.
    - In order to retrieve every word from the string pool:

        - Create a variable that records the current start index of the word. Initially, this variable will be 0, let's call this variable `int curStart = 0`.
        - Create a loop that will loop through the array that store the length of each word, for each iteration `i`:
            - Create a char array that will store the character of the word, with the length of the current item in the length array.
            - Loop through this char array to construct the word, with the starting index as 0 (`int j = 0`), and the end index as the length of this char array. For each iteration `j`:
                - Get the character in the string pool by index. This index will be calculated by taking the summing `curStart` and `j` (`stringpool[curStart + j]`)
            - Update `curStart` by incrementing it with the length of the current word. (`curStart += length[i]`])

- **Alternative**: I would use `std::vector` instead if STL data structure is allowed so that I do not have to worry too much about memory allocation of the input data (`std::vector` will resize itself, preventing not allocating enough memory to the array).

Linked List

- I used another Linked List to connect the Key Value entries of the Hash Map. So that I can iterate through this Linked List Iterator, and get all the map entries, without the need of looping through the unused buckets of the hashmap.

5/ Algorithms used:

- Sorting Algorithm: `Merge Sort`
    - `Merge Sort` is a very well-performing divide-and-conquer algorithm. I used `bubble sort` at first due to its simple implementation. Then I optimised the program by implementing `merge sort` instead, improving the time complexity from `O(n^2)` to `O(nlog(n))`. Due to its nature
- djb2 hash function for the hashmap

- According to the Lassonde School of Engineers (no date), the author stated that a djb2 is a robust string hash function. Moreover, based on my test on different hash functions such as djb2, modular hashing, and the “lose lose” hash function on the `sample-long.txt` , I found that djb2 result in the least collision compare to others.

6/ Demo of working program:

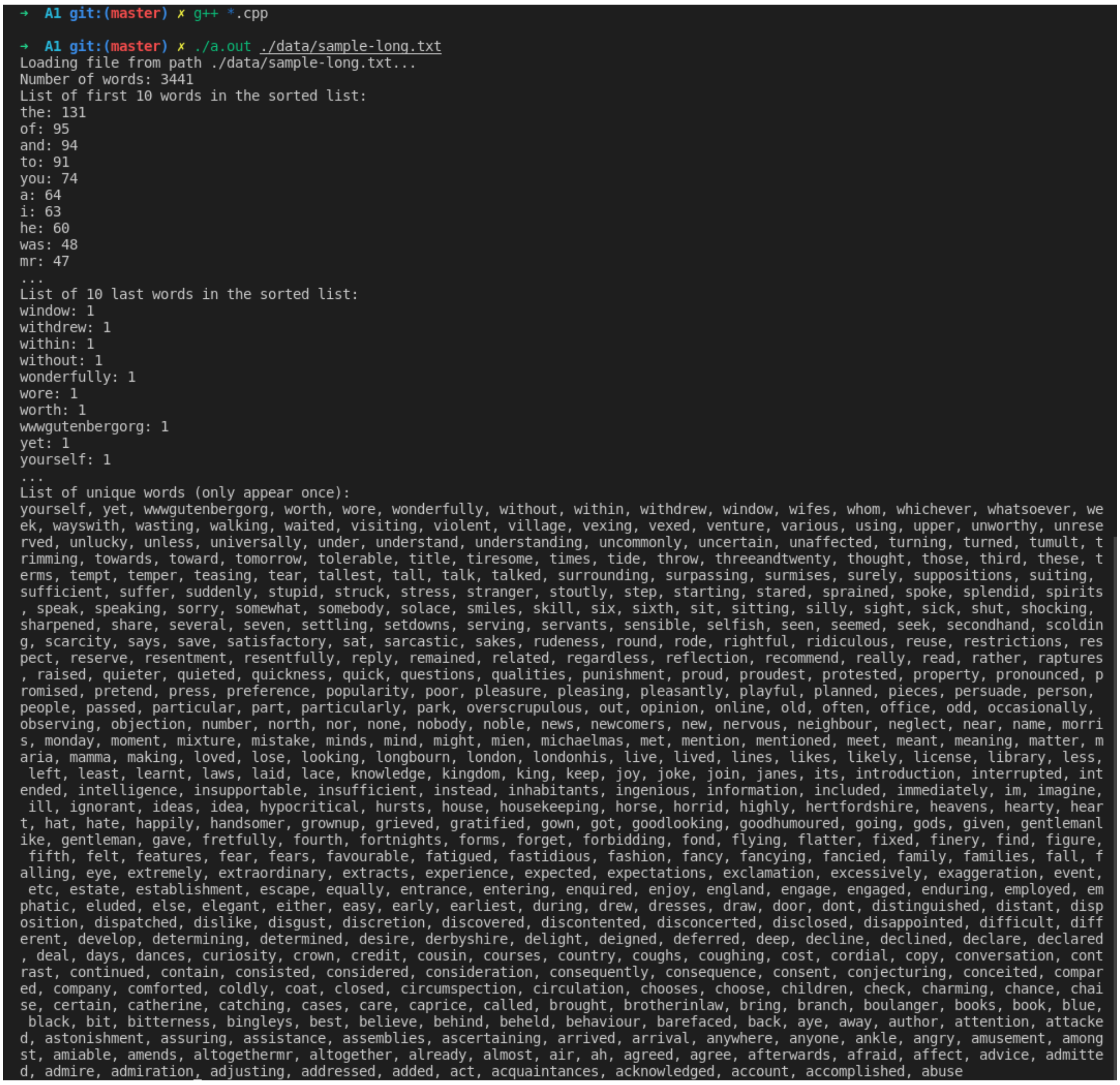
Compile the program:

> g++ *.cpp

Execute the program:

> ./a.out <path to text file>

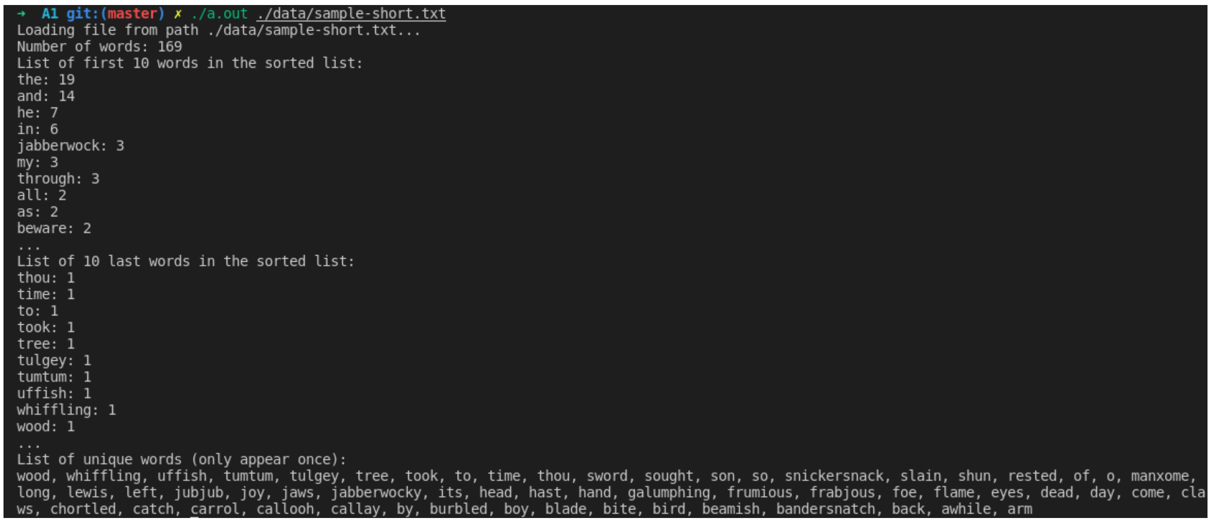
Execution toward sample-long.txt file:



```
➜  A1 git:(master) ✗ g++ *.cpp

➜  A1 git:(master) ✗ ./a.out ./data/sample-long.txt
Loading file from path ./data/sample-long.txt...
Number of words: 3441
List of first 10 words in the sorted list:
the: 131
of: 95
and: 94
to: 91
you: 74
a: 64
i: 63
he: 60
was: 48
mr: 47
...
List of 10 last words in the sorted list:
window: 1
withdrew: 1
within: 1
without: 1
wonderfully: 1
wore: 1
worth: 1
wwwgutenbergorg: 1
yet: 1
yourself: 1
...
List of unique words (only appear once):
yourself, yet, wwwgutenbergorg, worth, wore, wonderfully, without, within, withdrew, window, wifes, whom, whichever, whatsoever, we
ek, wayswith, wasting, walking, waited, visiting, violent, village, vexing, vexed, venture, various, using, upper, unworthy, unrese
rved, unlucky, unless, universally, under, understand, understanding, uncommonly, uncertain, unaffected, turning, turned, tumult, t
rimming, towards, toward, tomorrow, tolerable, title, tiresome, times, tide, throw, threeandtwenty, thought, those, third, these, t
erms, tempt, temper, teasing, tear, tallest, tall, talk, talked, surrounding, surpassing, surmises, surely, suppositions, suiting,
sufficient, suffer, suddenly, stupid, struck, stress, stranger, stoutly, step, starting, stared, sprained, spoke, splendid, spirits
, speak, speaking, sorry, somewhat, somebody, solace, smiles, skill, six, sixth, sit, sitting, silly, sight, sick, shut, shocking,
sharpened, share, several, seven, settling, setdowns, serving, servants, sensible, selfish, seen, seemed, seek, secondhand, scoldin
g, scarcity, says, save, satisfactory, sat, sarcastic, sakes, rudeness, round, rode, rightful, ridiculous, reuse, restrictions, res
pect, reserve, resentment, resentfully, reply, remained, related, regardless, reflection, recommend, really, read, rather, raptures
, raised, quieter, quieted, quickness, quick, questions, qualities, punishment, proud, proudest, protested, property, pronounced, p
romised, pretend, press, preference, popularity, poor, pleasure, pleasing, pleasantly, playful, planned, pieces, persuade, person,
people, passed, particular, part, particularly, park, overscrupulous, out, opinion, online, old, often, office, odd, occasionally,
observing, objection, number, north, nor, none, nobody, noble, news, newcomers, new, nervous, neighbour, neglect, near, name, morri
s, monday, moment, mixture, mistake, minds, mind, might, mien, michaelmas, met, mention, mentioned, meet, meant, meaning, matter, m
aria, mamma, making, loved, lose, looking, longbourn, london, londonhis, live, lived, lines, likes, likely, license, library, less,
 left, least, learnt, laws, laid, lace, knowledge, kingdom, king, keep, joy, joke, join, janes, its, introduction, interrupted, int
ended, intelligence, insupportable, insufficient, instead, inhabitants, ingenious, information, included, immediately, im, imagine,
 ill, ignorant, ideas, idea, hypocritical, hursts, house, housekeeping, horse, horrid, highly, hertfordshire, heavens, hearty, hear
t, hat, hate, happily, handsomer, grownup, grieved, gratified, gown, got, goodlooking, goodhumoured, going, gods, given, gentlemanl
ike, gentleman, gave, fretfully, fourth, fortnights, forms, forget, forbidding, fond, flying, flatter, fixed, finery, find, figure,
 fifth, felt, features, fear, fears, favourable, fatigued, fastidious, fashion, fancy, fancying, fancied, family, families, fall, f
alling, eye, extremely, extraordinary, extracts, experience, expected, expectations, exclamation, excessively, exaggeration, event,
 etc, estate, establishment, escape, equally, entrance, entering, enquired, enjoy, england, engage, engaged, enduring, employed, em
phatic, eluded, else, elegant, either, easy, early, earliest, during, drew, dresses, draw, door, dont, distinguished, distant, disp
osition, dispatched, dislike, disgust, discretion, discovered, discontented, disconcerted, disclosed, disappointed, difficult, diff
erent, develop, determining, determined, desire, derbyshire, delight, deigned, deferred, deep, decline, declined, declare, declared
, deal, days, dances, curiosity, crown, credit, cousin, courses, country, coughs, coughing, cost, cordial, copy, conversation, cont
rast, continued, contain, consisted, considered, consideration, consequently, consequence, consent, conjecturing, conceited, compar
ed, company, comforted, coldly, coat, closed, circumspection, circulation, chooses, choose, children, check, charming, chance, chai
se, certain, catherine, catching, cases, care, caprice, called, brought, brotherinlaw, bring, branch, boulanger, books, book, blue,
 black, bit, bitterness, bingleys, best, believe, behind, beheld, behaviour, barefaced, back, aye, away, author, attention, attacke
d, astonishment, assuring, assistance, assemblies, ascertaining, arrived, arrival, anywhere, anyone, ankle, angry, amusement, among
st, amiable, amends, altogethermr, altogether, already, almost, air, ah, agreed, agree, afterwards, afraid, affect, advice, admitte
d, admire, admiration, adjusting, addressed, added, act, acquaintances, acknowledged, account, accomplished, abuse
```

Execution toward sample-short.txt file:



```
➜  A1 git:(master) ✗ ./a.out ./data/sample-short.txt
Loading file from path ./data/sample-short.txt...
Number of words: 169
List of first 10 words in the sorted list:
the: 19
and: 14
he: 7
in: 6
jabberwock: 3
my: 3
through: 3
all: 2
as: 2
beware: 2
...
List of 10 last words in the sorted list:
thou: 1
time: 1
to: 1
took: 1
tree: 1
tulgey: 1
tumtum: 1
uffish: 1
whiffling: 1
wood: 1
...
List of unique words (only appear once):
wood, whiffling, uffish, tumtum, tulgey, tree, took, to, time, thou, sword, sought, son, so, snickersnack, slain, shun, rested, of, o, manxome,
long, lewis, left, jubjub, joy, jaws, jabberwocky, its, head, hast, hand, galumphing, frumious, frabjous, foe, flame, eyes, dead, day, come, cla
ws, chortled, catch, carrol, callooh, callay, by, burbled, boy, blade, bite, bird, beamish, bandersnatch, back, awhile, arm
```

7/ References:

Lassonde School of Engineer (no date), *Hash Functions,*
http://www.cse.yorku.ca/~oz/hash.html