CSCI-2270
Assignment 7
Instructor: Boese

Word Analysis

HW 2 Converted to use a Hash Table with Separate Chaining

Objectives

- Use a hashing function
- Store data in a chained hash table
- Search for data in a hash table

Background

The background idea is the same as Homework 2. This time you will implement a hash table using separate chaining with linked lists.

What your program needs to do

Use the text files from HW 7, **NOT** HW 2

- HW7-HungerGames_edit.txt
- HW7-stopWords.txt

Your class will be able to calculate the following information on any text file:

- The top n words (excluding stop words; n is also a command-line argument) and the number of times each word was found
- The total number of unique words (excluding stop words) in the file
- The total number of words (excluding stop words) in the file

Example:
Your program takes four command-line arguments: the number of most common words to print out, the name of the text file to process, the stop word list file, and the size of your hash table.

Running your program using:

```
./a.out 10 HW7-HungerGames_edit.txt HW7-stopWords.txt 53
```

would return the 10 most common words in the file HW7-HungerGames_edit.txt, use a hash table of size 53, and should produce the following results:

```
682 - is
492 - peeta
479 - its
431 - im
427 - can
```

```
414 - says
379 - him
368 - when
367 - no
356 - are
#
Number of collisions: 7628
#
Unique non-stop words: 7681
#
Total non-stop words: 59045
```

Program Specifications

The following are requirements for your program:

- Your Hash Table must be **implemented in a class** based on the provided HashTable.hpp.
- You must implement a driver program that utilizes your HashTable class. This driver program will contain your **main** function.
- When executing your driver program:
    - read in the number of most common words to process from the *first* command-line argument.
    - Read in the name of the text file to process from the *second* command-line argument.
    - Read stopwords from the file specified by the *third* command-line argument.
    - Create a hash table of size specified by the *fourth* command-line argument.
- Write a member function named **getStopwords** that takes the name of the stopwords file, fills the data member vector `vecIgnoreWords` with the stopwords, and returns void. Read in the file for a list of the top 50 most common words to ignore (e.g., Table 1).
    - The file will have one word per line, and always have exactly 50 words in the file. *We will test with files having different words in it!*
    - Your function will update the vector with a list of the words from the file.
    - *You can use your* **getStopWords** *function from Homework 2 as a starting point!*

- Implement a hashing function in member function **getHash** that will minimize collisions. There are many ways to do this, but for this assignment, we will use a hashing function known as *DJB2*. Pseudocode for this function is as follows:

```
int hash(string word)
    int hash = 5381;
    for each character c in word:
           hash = hash*33 + c
   hash = hash % hashTableSize
   if(hash < 0) hash+=hashTableSize
    return hash;
```

A detailed explanation of why this hash function is suitable for hashing strings can be found on the web. It is easy to see, however, why an overly simple hash function, such as summing up the ASCII values of the characters in a string, may not minimize collisions. A simple sum of ASCII values would result in a collision between all words made up of the same characters (cat and act, for example). The DJB2 hash avoids this by applying the multiplication step as it iterates through the characters in the string. However, due to repeated multiplication, it is possible that the hash value will undergo integer overflow. That is the reason for the check to see if the hash value is less than 0. Feel free to experiment with different hash functions and hash table sizes and see what happens to the number of collisions, but use the DJB2 hash for your submission.

- Store the unique words found in the file that are not in the stopword list in a hash table.
    - Check if the word is a stopword first, and if it is, then ignore that word.
    - Use the hashing function to determine where in the hash table the word should be stored.
    - Search the linked list at that location in the hash table array for the word.
        - If the list does not exist yet, dynamically allocate a new struct, make this new struct the head of the linked list, and add 1 to the number of unique words.
        - If the word is present in the list, add one to the count.
        - If the word is not present,
            - dynamically allocate a new struct and add it to the list
            - Add one to the number of unique words
            - If there was already something in this spot in the hash table, add 1 to the number of collisions.
    - This method of using linked lists to deal with collisions is called Separate Chaining with Linked Lists.
    - Hint: in your **main** function, after you have read in a word from the text file, this can be easily accomplished by using the **isStopWord**,

isInTable, **incrementCount**, and **addWord** member functions described below.

- Before your program ends, be sure to free all dynamically allocated memory via the class destructor.
- Implement a member function **isStopWord** that takes a string as an argument and returns true if the string is a stopword, or returns false otherwise.
- Implement a member function **isInTable** that takes a string as an argument and returns true if the string is already stored in the hash table, or returns false otherwise.
- Implement a member function **incrementCount** that increments the count of a word already stored in the hash table by 1.
- Write a member function named **addWord** that creates a new wordItem struct and adds it to the hash table at the appropriate location.
- Write a member function named **searchTable** that takes a string as an argument and returns a pointer to the wordItem struct that stores the string, or NULL if the string is not currently stored in the hash table.
- Implement getter methods for *numCollisions* and *numUniqueWords*.
- Output the top n most frequent words, number of collisions, number of unique non-stop words, and total non-stop words
    - Write a member function named **printTopN** that takes the value of N as an argument and determines the top N words in the array. *Hint: Declare an array of pointers of size n (static declaration), and use the insertIntoSortedArray algorithm from Assignment 1 to fill this array with words with the largest counts.*
    - Then print out the top N words using the specified output.
    - Print other output as specified elsewhere in this write-up (number of collisions, number of unique non-stop words, and total number of non-stop words.)
- Format your output the following way *(and reference the example above).*
    - When you output the top n words in the file, the output needs to be in order, with the most frequent word printed first. The format for the output needs to be:

```
Count - Word
#
Number of collisions: <number of collisions>
#
Unique non-stop words: <number of unique words>
#
Total non-stop words: <total number of words>
```

- Generate the output with these commands:

in **printTopN**:

```
cout << wordItem.count << " – " << wordItem.word << endl;
```

in **main:**

```
cout << "#" << endl;
cout << "Number of collisions: " << myHashTable.getNumCollisions() << endl;
cout << "#"<<endl;
cout << "Unique non-stop words: " << myHashTable.getNumUniqueWords() << endl;
cout << "#" << endl;
cout << "Total non-stop words: " << myHashTable.getTotalNumberNonStopWords() << endl;
```

- Include a comment block at the top of the .cpp files with your name, assignment number, date and course instructor.
- Make sure your code is commented enough to describe what it is doing.

Table 1. Top 50 most common words in the English language

Rank	Word	Rank	Word	Rank	Word
1	The	18	You	35	One
2	Be	19	Do	36	All
3	To	20	At	37	Would
4	Of	21	This	38	There
5	And	22	But	39	Their
6	A	23	His	40	What
7	In	24	By	41	So
8	That	25	From	42	Up
9	Have	26	They	43	Out
10	I	27	We	44	If
11	It	28	Say	45	About
12	For	29	Her	46	Who
13	Not	30	She	47	Get
14	On	31	Or	48	Which
15	With	32	An	49	Go
16	He	33	Will	50	Me
17	As	34	My		

- Use the following code for the HashTable class header file(type it in so you get more practice setting up classes):

HashTable.hpp

```
#ifndef HW_7_HASH_TABLE
#define HW_7_HASH_TABLE

#include <string>
#include <vector>


// struct to store word + count combinations
```

```
struct wordItem
{
    std::string word;
    int count;
    wordItem* next;
};

const int STOPWORD_LIST_SIZE = 50;

class HashTable{
public:
        HashTable(int hashTableSize);
        ~HashTable();
        void getStopWords(char *ignoreWordFileName);
        bool isStopWord(std::string word);
        bool isInTable(std::string word);
        void incrementCount(std::string word);
        void addWord(std::string word);
        int getTotalNumberNonStopWords();
        void printTopN(int n);
        int getNumUniqueWords();
        int getNumCollisions();
private:
        int getHash(std::string word);
        wordItem* searchTable(std::string word);
        int numUniqueWords;
        int numCollisions;
        int hashTableSize;
        wordItem** hashTable;
        std::vector<std::string> vecIgnoreWords =
                std::vector<std::string>(STOPWORD_LIST_SIZE);
};

#endif
```

Submitting Your Code:
Log into Moodle and go to the Homework 7 link. It is set up in the quiz format. Follow the instructions on each question to submit all or parts of each assignment question. You can check your solution to each question by clicking on the "Check" button. Note that you can only submit your homework **once**.

Note: there is no late period on assignments! If you miss the deadline or do not do well, you can sign up for an optional grading interview to get up to half the points missed

back. There is also an optional extra credit assignment at the end of the semester you can use to replace one of your homework scores.