```
/* ======================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: hash.h
 * ======================================================*/

#ifndef HASH_H
#define HASH_H

#include "list.h"

template <class KeyType>
class HashTable;

template <class KeyType>
std::ostream& operator<<(std::ostream& os, const HashTable<KeyType>& hash);

template <class KeyType>
class HashTable
{
        public:
                HashTable(int numSlots);
                ~HashTable();

                KeyType* get(KeyType& k);
                void insert(KeyType *k);
                void remove(KeyType& k);

                std::string toString(int slot);

        protected:
                int count;
                int slots;
                List<KeyType> *table;  // an array of List<KeyType*>'s

                friend std::ostream& operator<< <KeyType>(std::ostream& os, const HashTable<Ke
yType>& hash);
};

class KeyError { };

#include "hash.cpp"

#endif
```

```
/* ========================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: hash.cpp
 * ========================================================*/

#include <sstream>
using namespace std;


/* ========================================================
 *  Function: Constructor
 *
 *  Precondition: A HashTable is declared with a specific number of slots determined
 *  by a integer parameter value.
 *
 *  Postcondition: A HashTable is created with "numSlots" amount of slots that contain
 *  a list within them.
 * ========================================================*/
template <class KeyType>
HashTable<KeyType>::HashTable(int numSlots)
{
        table = new List<KeyType>[numSlots];
        slots = numSlots;
        count = 0;
}

/* ========================================================
 *  Function: Destructor
 *
 *  Precondition: A HashTable has been initialized.
 *
 *  Postcondition: Deletes the table pointer.
 * ========================================================*/
template <class KeyType>
HashTable<KeyType>::~HashTable()
{
        delete [] table;
}

/* ========================================================
 *  Function: get
 *
 *  Precondition: A key of the type KeyType is passed in as a parameter in order to find
 *  the corresponding value.
 *
 *  Postcondition: Either the key's corresponding value is returned or a KeyError is thrown
 *  meaning that there is no such key in the HashTable.
 * ========================================================*/
template <class KeyType>
KeyType* HashTable<KeyType>::get(KeyType& k)
{
        int s = k.hash(slots);
        int index = table[s].index(k);
        KeyType *x = table[s][index];
        if (index > -1)
                return x;
        else
                throw KeyError();
}

/* ========================================================
```

```
 *  Function: insert
 *
 *  Precondition: A key of the type KeyType is passed in as a parameter in order to insert a
 *  value in to the HashTable.
 *
 *  Postcondition: A value is added to the list of a certain slot calculated by the Hash funct
ion.
 * ========================================================*/
template <class KeyType>
void HashTable<KeyType>::insert(KeyType *k)
{
        int s = k->hash(slots);
        table[s].append(k);
        count++;
}

/* ========================================================
 *  Function: remove
 *
 *  Precondition: A key of the type KeyType is passed in as a parameter in order to find the
 *  corresponding value.
 *
 *  Postcondition: Once the value is found the value is removed completely from the HashTable.
 *  If the value is not found, then a Key Error is thrown, which means that no such value exis
ts.
 * ========================================================*/
template <class KeyType>
void HashTable<KeyType>::remove(KeyType& k)
{
        int s = k.hash(slots);
        int index = table[s].index(k);
        if (index > -1)
                table[s].remove(k);
        else
                throw KeyError();
        count--;
}

/* ========================================================
 *  Function: toString
 *
 *  Precondition: A HashTable is initialized.
 *
 *  Postcondition: Returns a string representation of the HashTable.
 * ========================================================*/
template <class KeyType>
std::string HashTable<KeyType>::toString(int slot)
{
        std::string str = "{";
        for(int i=0; i < table[slot].length() - 1; i++) {
                std::string result;
                std::ostringstream convert;
                convert<<table[slot][i]<<", ";
                result = convert.str();
                str = str + result;
        }

        if(table[slot].length() != 0) {
                std::string result;
                std::ostringstream convert;
                convert<<table[slot][table[slot].length() - 1];
                result = convert.str();
                str = str + result;
```

```
        }
        std::string result;
        std::ostringstream convert;
        convert<<"}";
        result = convert.str();
        str = str + result;
        return str;
}

/* =========================================================
 *  Function: << operator
 *
 *  Precondition: A HashTable is initialized.
 *
 *  Postcondition: Allows the ostream operator to print our a string representation of a
 *  HashTable.
 * =========================================================*/
template <class KeyType>
std::ostream& operator<<(std::ostream& os, const HashTable<KeyType>& hash)
{

        std::stringstream ss;

        bool first = true;
        ss << "{";
        for (int slot = 0; slot < hash.slots; slot++)
        {
                for (int index = 0; index < hash.table[slot].length(); index++)
                {
                        if (first){
                                first = false;
                        }
                        else{
                                ss << ", ";
                        }
                        ss << hash.table[slot][index];
                }
        }
        ss << "}";

        os << ss.str();
        return os;
}
```

```
/* ======================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: test_hash.cpp
 * ======================================================*/

#include "hash.h"
#include <iostream>
#include <cassert>
using namespace std;

class Test
{
        public:
                Test(int KeyValue);
                int hash(int slots) {return key % slots;}
                bool operator!=(const Test& otherTest) {return (key != otherTest.key);}
                bool operator==(const Test& otherTest) {return (key == otherTest.key);}
                int key;
                friend std::ostream& operator<< (std::ostream& os, const Test& t) {os << t.key
;  return os;}
};

Test::Test(int KeyValue)
{
        key = KeyValue;
}

void testConstructor()
{
        HashTable<Test> h(5);
}

void testInsert()
{
        HashTable<Test> h(5);
        Test t(5);
        Test *r = &t;
        h.insert(r);
        Test q(10);
        r = &q;
        h.insert(r);
        assert(h.toString(0) == "{5, 10}");
}

void testGet()
{
        HashTable<Test> h(5);
        Test t(5);
        Test *r = &t;
        h.insert(r);
        h.get(*r);
        try
        {
                assert(*(h.get(*r)) == 5);
        }
        catch (KeyError exception)
        {
                cerr<<"Error: Key does not exist. Cannot get."<<endl;
        }
}
```

```
void testRemove()
{
        HashTable<Test> h(5);
        Test t(5);
        Test *r = &t;
        h.insert(r);
        h.remove(*r);
        try
        {
                assert(h.toString(0) == "{}");
        }
        catch (KeyError exception)
        {
                cerr<<"Error: Key does not exist. Cannot remove."<< endl;
        }
}

int main()
{

        testConstructor();
        testInsert();
        testGet();
        testRemove();
        return 0;
}
```

```
/* ========================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: dict.h
 * ========================================================*/

#ifndef _DICT_H
#define _DICT_H

#include "hash.h"

template <class KeyType>
class Dictionary: public HashTable<KeyType>
{
        public:
                Dictionary(int numslots)
                : HashTable<KeyType>(numslots) {};

                bool empty() const;

        //private:
                int slots;
                int count;
};

#include "dict.cpp"

#endif
```

```
/* ========================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: dict.cpp
 * ========================================================*/
using namespace std;


/* ========================================================
 *  Function: empty
 *
 *  Precondition: A Dictionary is initialzed in order to check if it is empty
 *
 *  Postcondition: A boolean value is returned. True if the Dictionary's count is 0, and false
 if
 *  it is greater than 0.
 * ========================================================*/
template <class KeyType>
bool Dictionary<KeyType>::empty() const
{
        return count == 0;
}
```

```
/* ========================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: movie.h
 * ========================================================*/

#ifndef _MOVIE_H
#define _MOVIE_H

#include "list.h"
#include <iostream>
#include <sstream>

class Movie
{
        public:
                Movie(string xtitle="", string xcast="") {title = xtitle; cast = xcast;}

                int hash(int slots)
                {
                        int length = title.length(), val = 0;
                        for(int i = 0; i < length; i++) {
                                val = title[i] * title[i] + val * 19;
                        }
                        val = ((val % slots) + slots) % slots;          //converts val to posi
tive value if negative, does nothing if positive
                        return val;
                }
                Movie& operator=(const Movie& otherMovie) {title = otherMovie.title; cast = ot
herMovie.cast; return *this;}

                bool operator==(const Movie& otherMovie) {return (title == otherMovie.title);}
                bool operator!=(const Movie& otherMovie) {return !(*this == otherMovie);}

                friend std::ostream& operator<< (std::ostream& os, Movie& m) {os << m.cast<<en
dl; return os;}

                string title;
                string cast;
};

#endif
```

```
/* =========================================================
 *  Authors: Christopher Castillo & Cole Shaler
 *  Course: CS271, Spring 2016
 *  Date: March 28, 2016
 *  File: query_movies.cpp
 * =========================================================*/

#include "dict.h"
#include "movie.h"
#include "list.h"
#include <iostream>
#include <fstream>
using namespace std;

int main()
{
        Dictionary<Movie> movies(1000);
        ifstream infile;
        infile.open("movies.txt");
        int lines = 1;


        string line;
        string *c;

        while(getline(infile, line))
        {
                Movie *movie = new Movie;
                int length = line.length();
                string titles;

                titles = line.substr(0,line.find('\t',0));                              //Star
ts from 0, then gets the tab
                                                                                position

                int start = line.find('\t',0)+1;
//Gets position after the title

                string castList = line.substr(start,length - start);    //Starts after title a
nd gets what
                                                        remains in line

                movie->title = titles;
                movie->cast = castList;

                movies.insert(movie);
        }

        infile.close();

        string film;
        cout << "Please select a movie you would like to see the cast for: ";
        getline(cin, film);

        while(film != "q") {
                Movie *movie = new Movie;
                movie->title = film;
                Movie x = *(movies.get(*movie));
                cout << x << endl;
                cout << "Please select a movie you would like to see the cast for, or enter q
to quit: ";
                getline(cin, film);
        }
```

```
        return 0;
}
```

CS 271 - Data Structures
Project #5

Christopher Castillo and Cole Shaler

March 30, 2016

Problem 1. Develop a good hash function mapping strings to integers in $\{0, 1, ..., m-1\}$. Develop this hash function on your own, without consulting external sources. Evaluate your hash function by writing a small program that calls your hash function on every word in the dictionary file /usr/share/dict/words. (Note that you do not need to actually insert the words into a hash table.) Create a histogram that displays how evenly your hash function assigns words to slots when m = 1000. Also compute the average number of words assigned to a slot.

Explanation. The hash function uses the length of a word in order to iterate through each character. The ASCII value of each character is squared and added to 19 times the sum so far. The final value is then modded by the number of slots, which in this case is 1000. The mean is 99.171, the variance is 109.434, and the standard deviation is 10.461. □