```
#include "hashTable.h"
#include<iostream>
#include<climits>
#include<fstream>
#include<vector>
#include<string>

using namespace std;



void testHash(){
        cout << "Test Hash Function" << endl;
        int numSlots = 13001; //prime
        str s1("Tung Luu");
        assert(s1.hash(numSlots) == 1821);
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        str s2("Wilson Le");
        assert(s2.hash(numSlots) == 3402);
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
}

void testHtTostring(){
        cout << "Test Hash Table toString" << endl;
        int numSlots = 13001; //prime
        HashTable<Data<str, int> > ht(numSlots);
        assert(ht.toString(0) == "[]");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        assert(ht.toString(2) == "[]");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        assert(ht.toString(13001) == "[]");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
}

void testHtConstructor(){
        cout << "Test Hash Table Constructor" << endl;
        int numSlots1 = 13001; //prime
        HashTable<Data<str, int> > ht1(numSlots1);
        assert(ht1.slots == numSlots1);
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        assert(ht1.table->length() == 0);
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        assert(ht1.table->toString() == "[]");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
        int numSlots2 = 13381; //prime
        HashTable<Data<str, int> > ht2(numSlots2);
        assert(ht2.slots == numSlots2);
        cout << "\tTest 4 Passed â\234\224ï¸\216" << endl;
        assert(ht2.table->length() == 0);
        cout << "\tTest 5 Passed â\234\224ï¸\216" << endl;
        assert(ht2.table->toString() == "[]");
        cout << "\tTest 6 Passed â\234\224ï¸\216" << endl;
}

void testHtInsert(){
        cout << "Test Hash Table Insert" << endl;
        int numSlots = 13001; //prime
        HashTable<Data<str, string> > ht1(numSlots);
        Data<str, string> *d1 = new Data<str, string>(str("Hello"), "world");
        Data<str, string> *d2 = new Data<str, string>(str("CS271"), "Wilson Le");
        Data<str, string> *d3 = new Data<str, string>(str("CS271"), "Tung Luu");
        ht1.insert(d1);
        ht1.insert(d2);
        ht1.insert(d3);
        assert(ht1.toString(d1->hash(numSlots)) == "[world]");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        assert(ht1.toString(d2->hash(numSlots)) == "[Wilson Le, Tung Luu]");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        assert(ht1.toString(d3->hash(numSlots)) == "[Wilson Le, Tung Luu]");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
```

```
}

void testHtRemove(){
        cout << "Test Hash Table Remove" << endl;
        int numSlots = 13001; //prime
        HashTable<Data<str, string> > ht1(numSlots);
        Data<str, string> *d1 = new Data<str, string>(str("Hello"), "world");
        Data<str, string> *d2 = new Data<str, string>(str("CS271"), "Wilson Le");
        Data<str, string> *d3 = new Data<str, string>(str("CS271"), "Tung Luu");
        ht1.insert(d1);
        ht1.insert(d2);
        ht1.insert(d3);
        ht1.remove(*d1);
        assert(ht1.toString(d1->hash(numSlots)) == "[]");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        ht1.remove(*d2);
        assert(ht1.toString(d2->hash(numSlots)) == "[Tung Luu]");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        ht1.remove(*d3);
        assert(ht1.toString(d3->hash(numSlots)) == "[]");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
}

void testHtClone(){
        cout << "Test Hash Table Clone" << endl;
        int numSlots = 11; //prime
        HashTable<Data<str, string> > ht1(numSlots);
        Data<str, string> *d1 = new Data<str, string>(str("Hello"), "world");
        Data<str, string> *d2 = new Data<str, string>(str("CS271"), "Wilson Le");
        Data<str, string> *d3 = new Data<str, string>(str("CS271"), "Tung Luu");
        ht1.insert(d1);
        ht1.insert(d2);
        ht1.insert(d3);
        HashTable<Data<str, string> > ht2 = ht1;
        HashTable<Data<str, string> > ht3(ht1);
        ht2.remove(*d1);
        ht3.remove(*d2);
        assert(ht1.toString(d1->hash(numSlots)) == "[world]");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        assert(ht1.toString(d2->hash(numSlots)) == "[Wilson Le, Tung Luu]");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        assert(ht1.toString(d3->hash(numSlots)) == "[Wilson Le, Tung Luu]");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
        assert(ht2.toString(d1->hash(numSlots)) == "[]");
        cout << "\tTest 4 Passed â\234\224ï¸\216" << endl;
        assert(ht2.toString(d2->hash(numSlots)) == "[Wilson Le, Tung Luu]");
        cout << "\tTest 5 Passed â\234\224ï¸\216" << endl;
        assert(ht3.toString(d1->hash(numSlots)) == "[world]");
        cout << "\tTest 6 Passed â\234\224ï¸\216" << endl;
        assert(ht3.toString(d2->hash(numSlots)) == "[Tung Luu]");
        cout << "\tTest 7 Passed â\234\224ï¸\216" << endl;
}

int main(){
        testHash();
        testHtTostring();
        testHtConstructor();
        testHtInsert();
        testHtRemove();
        testHtClone();
        return 0;
}
```

```
#ifndef DICT_H
#define DICT_H
#include "hashTable.h"
#include "data.h"
template <class Key, class Value>
class Dict{
        friend void testDictDestructor();
        public:
                Dict();

                // insert data into list
                void insert(Data<Key, Value> *d);

                // get data from list
                Data<Key, Value> * get(const Data<Key, Value>& d) const;

                // remove data from list
                void remove(const Data<Key, Value>& d);

                // return string representation of dictionary
                string toString();

        private:
                HashTable<Data<Key, Value> > ht;
                int size;
};

#include "dict.cpp"
#endif
```

```
#include <sstream>
#include "hashTable.h"
#include "dict.h"
using namespace std;

template <class Key, class Value>
Dict<Key, Value>::Dict(){
        this->size = 13001;
        this->ht = HashTable<Data<Key, Value> >(size);
}

template <class Key, class Value>
void Dict<Key, Value>::insert(Data<Key, Value> *d){
        this->ht.insert(d);
}

template <class Key, class Value>
Data<Key, Value> * Dict<Key, Value>::get(const Data<Key, Value>& k) const {
    return this->ht.get(k);
}

template <class Key, class Value>
void Dict<Key, Value>::remove(const Data<Key, Value>& k){
        this->ht.remove(k);
}

template <class Key, class Value>
string Dict<Key, Value>::toString(){
    return ht.toString();
}
```

```
#include "dict.h"
#include<iostream>
#include<climits>
#include<fstream>
#include<vector>
#include<string>

using namespace std;

void testDictToString(){
        cout << "Test Dict toString" << endl;
        Dict<str, string> d;
        d.insert(new Data<str, string>(str("test"), "testValue1"));
        assert(d.toString() == "{\n[test: testValue1]\n}");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        d.insert(new Data<str, string>(str("test"), "testValue2"));
        assert(d.toString() == "{\n[test: testValue1, test: testValue2]\n}");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        d.insert(new Data<str, string>(str("test1"), "testValue2"));
        assert(d.toString() == "{\n[test1: testValue2]\n[test: testValue1, test: testValu
e2]\n}");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
}

void testDictInsert(){
        cout << "Test Dict Insert" << endl;
        Dict<str, string> d;
        d.insert(new Data<str, string>(str("test"), "testValue1"));
        assert(d.toString() == "{\n[test: testValue1]\n}");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        d.insert(new Data<str, string>(str("test"), "testValue2"));
        assert(d.toString() == "{\n[test: testValue1, test: testValue2]\n}");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
        d.insert(new Data<str, string>(str("test1"), "testValue2"));
        assert(d.toString() == "{\n[test1: testValue2]\n[test: testValue1, test: testValu
e2]\n}");
        cout << "\tTest 3 Passed â\234\224ï¸\216" << endl;
}

void testDictGet(){
        cout << "Test Dict Insert" << endl;
        Dict<str, string> d;
        d.insert(new Data<str, string>(str("A"), "hello"));
        assert(d.get(Data<str, string>(str("A"), ""))->toString() == "A: hello");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        d.insert(new Data<str, string>(str("B"), "world"));
        assert(d.get(Data<str, string>(str("B"), ""))->toString() == "B: world");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
}

void testDictRemove(){
        cout << "Test Dict Insert" << endl;
        Dict<str, string> d;
        d.insert(new Data<str, string>(str("A"), "hello"));
        d.insert(new Data<str, string>(str("A"), "helloo"));
        d.insert(new Data<str, string>(str("B"), "world"));
        d.insert(new Data<str, string>(str("B"), "worldd"));
        d.remove(Data<str, string>(str("A"), ""));
        d.remove(Data<str, string>(str("B"), ""));
        assert(d.get(Data<str, string>(str("A"), ""))->toString() == "A: helloo");
        cout << "\tTest 1 Passed â\234\224ï¸\216" << endl;
        assert(d.get(Data<str, string>(str("B"), ""))->toString() == "B: worldd");
        cout << "\tTest 2 Passed â\234\224ï¸\216" << endl;
}

int main(){
        testDictToString();
        testDictInsert();
        testDictGet();
        testDictRemove();
```

```
        return 0;
}
```

```
#ifndef DATA_H
#define DATA_H
#include <string>
#include <iostream>
#include <climits>

// class str is a string that has a hash function
class str {
        public:
                str(){
                        this->data = "";
                }
                str(string s){
                        this->data = s;
                }
                int hash(int numSlots) const{
                        // The maximum value of numSlots should be INT_MAX
                        numSlots = min(numSlots, INT_MAX);

                        // Since the char type has 8-bits, which can represents 256 chara
cters,
                        // we will choose 257 as our base because 257 is a prime which is
 close to 256.
                        int base = 257;
                        int powOfBase = 1;
                        int mod = numSlots;

                        int hashRes = 0; // The hash result
                        for(int i = 0; i < this->data.length(); i++) {
                                unsigned char c = this->data[i];
                                // Using the property (a+b) mod n=(a mod n + b mod n) mod
 n
                                // and the property (ab) mod p = ( (a mod p) (b mod p) )
mod p,
                                // we can use the following formulas to calculate the sam
e result as
                                // (hashing a string by expressing it as radix-257 and th
en mod by a number)
                                hashRes = (hashRes + (c * powOfBase)) % mod;
                                powOfBase = (powOfBase * base) % mod;
                        }

                        return hashRes;
                }
                friend ostream& operator<<(ostream& os, const str& s){
                        os << s.data;
                        return os;
                }
                friend bool operator<(const str& s1, const str& s2){
                        return s1.data < s2.data;
                }
                friend bool operator>(const str& s1, const str& s2){
                        return s1.data > s2.data;
                }
                friend bool operator==(const str& s1, const str& s2){
                        return s1.data == s2.data;
                }
        private:
                string data;
};

template <class K, class V>
class Data {
        public:
                Data(K k, V v){
                        this->key = k;
                        this->value = v;
                };
                int hash(int numSlots) const{
                        return this->key.hash(numSlots);
```

```
                }
        string toString() const {
            stringstream ss;
            ss << this->key << ": " << this->value;
            return ss.str();
        }
                friend ostream & operator<< (ostream& output, const Data &D ){
                        output << D.toString();
                        return output;
                }
                friend bool operator<(const Data& Data1, const Data& Data2){
                        return Data1.key < Data2.key;
                }
                friend bool operator>(const Data& Data1, const Data& Data2){
                        return Data1.key > Data2.key;
                }
                friend bool operator==(const Data& Data1, const Data& Data2){
                        return Data1.key == Data2.key;
                }
        private:
                K key;
                V value;
};
#endif
```

```
#include<iostream>
#include <fstream>
#include <string>
#include <sys/time.h>
#include "dict.h"
#include "data.h"
using namespace std;

int main()
{
        Dict<str, string> movieDict;
        ifstream movieData("movies_mpaa.txt");

        if(movieData.is_open()) {
                string movie;
                // start timer
                timeval timeBefore, timeAfter;
                long diffSeconds, diffUSeconds;
                gettimeofday(&timeBefore, NULL);

                while(getline(movieData, movie)){
                        int titleEnd = movie.find("\t");
                        string title = movie.substr(0, titleEnd);
                        string cast = movie.substr(titleEnd + 1, movie.size() - titleEnd
- 1);
                        Data<str, string>* movie = new Data<str, string>(str(title), cast
);
                        movieDict.insert(movie);
                }
                movieData.close();

                // stop timer
                gettimeofday(&timeAfter, NULL);
                diffSeconds = timeAfter.tv_sec - timeBefore.tv_sec;
                diffUSeconds = timeAfter.tv_usec - timeBefore.tv_usec;
                cout << diffSeconds + diffUSeconds/1000000.0 << endl;

        }
        string title = "\"A Woman of Independent Means\" (1995)";
        Data<str, string>* sample = new Data<str, string>(str(title), "");
        Data<str, string>* movie = movieDict.get(*sample);
    cout<< *movie << endl;
}
```