```
#ifndef HASH_H
#define HASH_H

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

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);

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

#include "hash.cpp"

#endif
```

```
#include <iostream>
#include <cstdlib>
#include <string>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include "hash.h"
using namespace std;




/*
 *              Construct a hashtable with slots equal to numSlots.
 *              Precondition: numSlots > 0
 *              Postcondition: Constructs a new Hashtable holding numSlots
 */
template <class KeyType>
HashTable<KeyType>::HashTable(int numSlots)
{
        slots = numSlots;
        table = new List<KeyType>[numSlots];
}

/*
 *              Destructor
 *              Precondition:
 *              Postcondition: table is deleted
 */
template <class KeyType>
HashTable<KeyType>::~HashTable()
{

        delete[] table;
}

/*
 *              Inserts a KeyType into the HashTable
 *              Precondition: k is a non-null object
 *              Postcondition: k is added to the hash table
 */
template <class KeyType>
void HashTable<KeyType>::insert(KeyType *k)
{
        int slot = k->hash(slots);
        table[slot].insert(0,k);
}

/*
 *              Get the value associated with k
 *              Precondition: k is a non-null object
 *              Postcondition: Returns the value associated with k, or null if not found
 */
template <class KeyType>
KeyType* HashTable<KeyType>::get(KeyType& k)
{
        int slot = k.hash(slots);
        int ind = (table[slot]).index(k);

        //throw error if k not found
        return (table[slot])[ind];

}
```

```
/*
 *              Removes a value from the hash table
 *              Precondition: k is a non-null object
 *              Postcondition: Remove k from the hashtable
 */
template <class KeyType>
void HashTable<KeyType>::remove(KeyType &k)
{
        //throw error if empty?

        table[k.hash(slots)].remove(k);
}

/*
 *              Generates a string representation of the hash table
 *              Precondition:
 *              Postcondition: Returns a string representation of the hash table
 */
template <class KeyType>
std::string HashTable<KeyType>::toString(int slot)
{

        string s = "(";

        for (int i = 0; i < table[slot].length(); i++)
        {
                stringstream str;
                string ss;
                s += "[";
                str << *((table[slot])[i]);
                str >> ss;
                s += (ss + "], ");
        }
        cout << s << "}" << endl;
        return s + "}";

}
```

```
#include <stdlib.h>
#include <fstream>
#include <cstdlib>
#include <string>
#include <sstream>
#include <stdio.h>
#include <stdlib.h>
#include "test.h"

using namespace std;

int main()
{

        Test T1, T2, T3, T4;
        T1.key = 1;             // 1
        T2.key = 2;             // 2
        T3.key = 3;             // 0
        T4.key = 4;             // 1



        //make hash table (&T1)
        HashTable<Test> *H1 = new HashTable<Test>(3);

        H1->insert(&T1);
        H1->insert(&T2);
        H1->insert(&T3);
        H1->insert(&T4);
        H1->remove(T1);
        H1->get(T2);



}
```

```
#ifndef _HASHDICTIONARY_H_
#define _HASHDICTIONARY_H_
#include <iostream>
#include <cstdlib>
#include "hash.h"

template <class KeyType>
class HashDictionary
{
public:
    int max_size;
    HashDictionary();
    ~HashDictionary();
    KeyType* get(KeyType& k);
    void insert(KeyType *x);
    void remove(KeyType& k);
    bool empty();
    int hash(string str); //should be in the DictionaryTest



private:

        HashTable<KeyType> *table;
};

#include "dict.cpp"

#endif // _HASHDICTIONARY_H_
```

```
#include <iostream>
#include <cstdlib>
#include "hash.h"


using namespace std;

template <class KeyType>
int HashDictionary<KeyType>::hash(string str)
{
   /* int hash = 5381;
    int c;
    const char *chr = str.c_str();

    while (c = *chr++)
        hash = ((hash << 5) + hash) + c;

    return abs(hash/10000000);*/

        int numslots = 1000;
        int len = str.length();
        unsigned int t = 0;


        for (int i = 0; i < len; i++)
        {
                t = ((25 * t + str[i]) *str[0] + (len/2) % len) % numslots;
        }
        return t;

}

template <class KeyType>
HashDictionary<KeyType>::HashDictionary()
{
    max_size = 1000;
    table = new HashTable<KeyType>(max_size);
}

template <class KeyType>
HashDictionary<KeyType>::~HashDictionary()
{
    table.~HashTable();
}

template <class KeyType>
KeyType *HashDictionary<KeyType>::get(KeyType& k)
{
    int index = hash(k.title);
    return table->get(k);
}

template <class KeyType>
bool HashDictionary<KeyType>::empty()
{
    if (table[0] == 0)
        return true;
    else
        return false;
}

template <class KeyType>
void HashDictionary<KeyType>::insert(KeyType *x)
```

```
{
    int index = hash(x->title);
    table->insert(x);
}

template <class KeyType>
void HashDictionary<KeyType>::remove(KeyType& k)
{
    int index = hash(k.title);
    table->remove(k);
}
```

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <stdio.h>
#include <sys/time.h>
#include <stdlib.h>
#include <string.h>
#include <sstream>

#include "dict.h"
using namespace std;
class Movie
{
public:
    string title;
    string year;
    string cast;



    Movie()
    {
        title = " ";
        cast = " ";
    }

    int hash (int numSlots)
        {
                        return (title.length() % numSlots);
        }

    Movie(string initTitle, string initCast)
    {
        title = initTitle;
        cast = initCast;
    }

    ~Movie()
    {

    };


    bool operator<(const Movie &other) const;
    bool operator==(const Movie &other) const;
    friend ostream &operator<<(ostream &strm, Movie &e);
};

bool Movie::operator==(const Movie &other) const
{
    return title == other.title;
}

bool Movie::operator<(const Movie &other) const
{
    return title < other.title;
}


ostream &operator<<(ostream &strm, Movie &e)
{
    strm << e.title;
    return strm;
```

```
}


int main()
{
    HashDictionary<Movie> *b = new HashDictionary<Movie>;
    Movie *movie;

    ifstream file("movies-mpaa.txt");
    string str;

   int start = clock();

    string mvieTitle;
    string theYear;


    while (getline(file, str))
    {
        Movie *e = new Movie();
        int j = 0;
        int k = 0;
        int l = 0;
        for (int i = 0; i < str.length(); i++)                          // This for loop gets
the title
        {
            if (str[i] == '(')
            {
                j = i-1;
                k = i;
                mvieTitle = str.substr(0, j);
                e->title = mvieTitle;
                break;
            }
        }

        for (int r = k; r < str.length(); r++)                          // Gets the year
        {
            if (str[r] == ')')
            {
                l = r;
                theYear = str.substr(k+1, 4);
                e->year = theYear;
                break;
            }

        }


        e->cast = str.substr(l+1, str.length()-l);
        b->insert(e);

    }

    int end = clock();
    cout << "it took " << end - start << " ticks, or " << ((float)end - start)/CLOCKS_PER_SEC
<< " seconds." << endl;


    bool done = false;
    int choice;
```

```
    while (!done)
    {
        cout << endl;
        cout << "MENU" << endl;
        cout << endl;
        cout << " 1. Search for a movie" << endl;
        cout << " 2. Delete a movie" << endl;
        cout << " 3. Insert a movie" << endl;
        cout << " 4. Quit" << endl;
        cout << endl;
        cout << "Choice: ";
        cin >> choice;

        cout << endl;
        if (choice == 1)
        {
            string movieTitle;
            cin.ignore(255, '\n');
            cout << "Enter the name of the movie: " << endl;
            getline(cin, movieTitle);
            movieTitle = movieTitle.substr(0, movieTitle.length());
            Movie *x = new Movie(movieTitle, " ");

            int start = clock();

            movie = b->get(*x);

            int end = clock();
            cout << "it took " << end - start << " ticks, or " << ((float)end - start)/CLOCKS_
PER_SEC << " seconds." << endl;


            if (movie != NULL)
            {
                cout << "Title: " << movie->title << '(' << movie->year << ')' << endl;
                cout << "Cast: " << movie->cast << endl;
            }
            else
            {
                cout << "not found" << endl;
            }

        }
        else if (choice == 2)
        {
            string movieTitle;
            cin.ignore(255, '\n');
            cout << "Title of Movie to Delete: ";
            getline(cin, movieTitle);
            Movie *x = new Movie(movieTitle, " ");

            timeval timeBefore, timeAfter;
            long diffSeconds, diffUSeconds;
            gettimeofday(&timeBefore, NULL);
            b->remove(*x);
            gettimeofday(&timeAfter, NULL);

            diffSeconds = timeAfter.tv_sec - timeBefore.tv_sec;
            diffUSeconds = timeAfter.tv_usec - timeBefore.tv_sec;
            cout << diffSeconds + diffUSeconds/1000000.0 << " seconds" << endl;

        }
        else if (choice == 3)
```

```
        {
            string movieTitle;
            string movieYear;
            string movieCast;
            cin.ignore(255, '\n');
            cout << "Title of movie to insert: ";
            getline(cin, movieTitle);
            cout << "Year of movie to insert: ";
            getline(cin, movieYear);
            cout << "Cast of movie to insert: ";
            getline(cin, movieCast);
            Movie *x = new Movie(movieTitle, movieCast);
            x->year = movieYear;
            b->insert(x);
        }
        else if (choice == 4)
        {
            done = true;
        }

        else
        {
                cout << "Please choose options 1, 2, 3, or 4." << endl;
                done = true;
        }
    }


}
```