```
//////////////////////////////////////////////////
//                                              //
//                                              //
//                  HashEntry.h                 //
//                                              //
//                                              //
//////////////////////////////////////////////////

#ifndef HASH_ENTRY
#define HASH_ENTRY

typedef int hashtype;
typedef int keytype;

/** The HashEntry class represents an entry in the hash table. A hash entry
holds both a key and the hash of that key **/

class HashEntry
{
      private:
            keytype key;
            hashtype hash;
      public:
            HashEntry(int key, int hash);//Constructor
            ~HashEntry();//Destructor
            keytype getKey();//Returns the key
            hashtype getHash();//Returns the hash
};

#endif
```

```
//////////////////////////////////////////////////
//                                              //
//                                              //
//                  HashEntry.cpp               //
//                                              //
```

```
//                                                  //
//////////////////////////////////////////////////////

#include "HashEntry.h"

/** The constructor for HashEntry
      @param the key to be stored
      @param the hash of the key
 **/
HashEntry::HashEntry(keytype key, hashtype hash)
{
      this->key = key;
      this->hash = hash;
}

/** The destructor for HashEntry **/
HashEntry::~HashEntry(){}

/** Returns the key of a hash entry
    @return The key
 **/
keytype HashEntry::getKey()
{
      return key;
}

/** Returns the hash of a hash entry
    @return the hash
 **/
hashtype HashEntry::getHash()
{
      return hash;
}
```

```
//////////////////////////////////////////////////////
//                                                  //
//                                                  //
//                  HashTable.h                     //
//                                                  //
//                                                  //
//////////////////////////////////////////////////////

#ifndef HASH_TABLE_H
#define HASH_TABLE_H

#define TABLE_SIZE 11//Max size of the hash table as well as modular
signature
```

```
#include <iostream>//Include for STL cout

#include "HashEntry.h"

/** The HashTable class represents the hash table **/

class HashTable
{
	private:
		HashEntry **table;//Array of entries in the table
		int size;//Holds the current size of the table
	public:
		HashTable();//Constructor
		~HashTable();//Destructor
		hashtype search(keytype key);//Returns the hash of a key
		void add(keytype key);//Adds a key to the hash table
		void printTable();//Uses STL to output the table contents
};

#endif
```

```
//////////////////////////////////////////////////
//                                              //
//                                              //
//                  HashTable.cpp               //
//                                              //
//                                              //
//////////////////////////////////////////////////

#include "HashTable.h"

/** Constructor for HashTable
 **/
HashTable::HashTable()
{
	table = new HashEntry*[TABLE_SIZE];
		for (int i = 0; i < TABLE_SIZE; i++)
			table[i] = NULL;
	size = 0;
}
```

```
/** Destructor for HashTable
 **/
HashTable::~HashTable()
{
      for (int i = 0; i < TABLE_SIZE; i++)
            if (table[i] != NULL)
                  delete table[i];
      delete[] table;
}

/** Returns the hash of a key if the key exists in the hash table
    @param the key we are searching for
    @return the hash of the key if it exists in the table
 **/
hashtype HashTable::search(keytype key)
{
      hashtype hash = (key % TABLE_SIZE);
      while (table[hash] != NULL && table[hash]->getKey() != key)
            hash = (hash + 1) % TABLE_SIZE;
      if (table[hash] == NULL)
            return -1;//Error, key not found
      else
            return table[hash]->getHash();
}

/** Adds a key to the hash table. Use linear probing if a collision occurs
    @param the key to add to the hash table
 **/
void HashTable::add(keytype key)
{
      if(size == TABLE_SIZE)
            return;//Hash table is full
      hashtype hash = (key % TABLE_SIZE);
      while (table[hash] != NULL && table[hash]->getKey() != key)
            hash = (hash + 1) % TABLE_SIZE;//Linear probing
      if (table[hash] != NULL)
            delete table[hash];
      table[hash] = new HashEntry(key, hash);
      size++;
}

/** Uses STL to output the contents of the hash table
 **/
void HashTable::printTable()
{
      for(int i = 0; i < TABLE_SIZE; i++;)
            if(table[i] != NULL]
                  std::cout << i << " : " << table[i]->getKey() << " : " <<
table[i]->getHash() << "\n";
            else
                  std::cout << i << "NULL : NULL\n";
}
```

```
//////////////////////////////////////////////////
//                                              //
//                                              //
//                  main.cpp                    //
//                                              //
//                                              //
//////////////////////////////////////////////////

#include <stdlib.h>//Included for STL rand()
#include <iostream>//Included for STL cout
#include <time.h>//Included for STL time()

#include "HashTable.h"

void Test1();
void Test2();
void Test3();

int main()
{
      srand(time(0));

      Test1();
      Test2();
      Test3();

      return 0;
}

/** Fill hash table with just a few keys and check the contents
```

```
 **/
void Test1()
{
	HashTable hashtable;

	for(int i = 0; i < 5; i++)
	{
		int key = rand() % 10 + 1;
		std::cout << "Adding " << key << " to hash table\n";
		hashtable.add(key);
	}

	std::cout << "Hash Table contents : \n\n";
	hashtable.printTable();
}

/** Fill hash table with too many keys and guarantee collisions will occur
then check contents
 **/
void Test2()
{
	HashTable hashtable;

	for(int i = 0; i < 20; i++)
	{
		int key = rand() % 3 + 1;
		std::cout << "Adding " << key << " to hash table\n";
		hashtable.add(key);
	}

	std::cout << "Hash Table contents : \n\n";
	hashtable.printTable();
}

/** Fill hash tables with a few known values and then search for them
 **/
void Test3()
{
	HashTable hashtable;

	int key1 = 5, key2 = 7, key3 = 13;

	std::cout << "Adding " << key1 << " to hash table\n";
	hashtable.add(key1);
	std::cout << "Adding " << key2 << " to hash table\n";
	hashtable.add(key2);
	std::cout << "Adding " << key3 << " to hash table\n";
	hashtable.add(key3);

	std::cout << "Hash Table contents : \n\n";
	hashtable.printTable();

	std::cout << "Searching for first key ..\n";
	int ret = hashtable.search(key1);
	if(ret != -1)
		std::cout << "First key found with hash " << ret << "\n";
```

```
	ret = -1;

	std::cout << "Searching for second key ..\n";
	int ret = hashtable.search(key2);
	if(ret != -1)
		std::cout << "Second key found with hash " << ret << "\n";
	ret = -1;

	std::cout << "Searching for third key ..\n";
	int ret = hashtable.search(key3);
	if(ret != -1)
		std::cout << "Third key found with hash " << ret << "\n";
}
```

Program Output :

```
Adding 7 to hash table
Adding 9 to hash table
Adding 3 to hash table
Adding 6 to hash table
Adding 2 to hash table
Hash table contents :

0 : 0 : 0
1 : 0 : 0
2 : 2 : 2
3 : 3 : 3
4 : 0 : 0
5 : 0 : 0
6 : 6 : 6
7 : 7 : 7
8 : 0 : 0
9 : 9 : 9
10 : 0 : 0
Adding 3 to hash table
Adding 2 to hash table
Adding 2 to hash table
Adding 2 to hash table
Adding 1 to hash table
Adding 3 to hash table
Adding 1 to hash table
Adding 2 to hash table
Adding 1 to hash table
Adding 1 to hash table
Adding 3 to hash table
Adding 2 to hash table
Adding 2 to hash table
Adding 1 to hash table
Adding 1 to hash table
Adding 3 to hash table
Adding 3 to hash table
Adding 1 to hash table
Adding 2 to hash table
```

Adding 1 to hash table
Hash table contents :

0 : 3 : 3
1 : 1 : 1
2 : 2 : 2
3 : 3 : 3
4 : 2 : 2
5 : 2 : 2
6 : 3 : 3
7 : 1 : 1
8 : 2 : 2
9 : 1 : 1
10 : 1 : 1
Adding 5 to hash table
Adding 7 to hash table
Adding 13 to hash table
Hash Table Contents :
0 : 0 : 0
1 : 0 : 0
2 : 13 : 2
3 : 0 : 0
4 : 0 : 0
5 : 5 : 5
6 : 0 : 0
7 : 7 : 7
8 : 0 : 0
9 : 0 : 0
10 : 0 : 0
Searching for first key ..
First key found with hash 5
Searching for second key ..
Second key found with hash 7
Searching for third key ..
Third key found with hash 2