Course: Principals of Software Development – ENSF 409
Lab 10
Instructor: M. Moshirpour
Student Name: Mitchell Sawatzky
Date Submitted: April 4, 2016

Exercise A

Iterator.cpp

```
#include <iostream>
#include <assert.h>
#include "mystring2.h"

using namespace std;

template <class T> class Vector {
public:
    class VectIter {
        friend class Vector;
        private:
            Vector *v; // points to a vector object of type T
            int index; // represents the subscript number of the vector's
                       // array.
        public:
        VectIter(Vector& x);

        T operator++();
        //PROMISES: increments the iterator's indes and return the
        //          value of the element at the index position. If
        //          index exceeds the size of the array it will
        //          be set to zero. Which means it will be circulated
        //          back to the first element of the vector.

        T  operator++(int);
        // PRIMISES: returns the value of the element at the index
        //           position, then increments the index. If
        //           index exceeds the size of the array it will
        //           be set to zero. Which means it will be circulated
        //           back to the first element of the vector.

        T  operator--();
        // PROMISES: decrements the iterator index, and return the
        //           the value of the element at the index. If
        //           index is less than zero it will be set to the
        //           last element in the aray. Which means it will be
        //           circulated to the last element of the vector.

        T  operator--(int);
```

```
        // PRIMISES: returns the value of the element at the index
        //           position, then decrements the index. If
        //           index is less than zero it will be set to the
        //           last element in the aray. Which means it will be
        //           circulated to the last element of the vector.

        T operator *();
        // PRIMISES: returns the value of the element at the current
        //           index position.

    };

    Vector(int sz);
    ~Vector();

    T& operator[](int i);
    // PRIMISES: returns existing value in the ith element of
    //           array or sets a new value to  the ith element in
    //           array.

    void ascending_sort();
    // PRIMISES: sorts the vector values in ascending order.

private:
    T *array;               // points to the first element of an array of T
    int size;                 // size of array
    void swap(T&, T&);   // swaps the values of two elements in array
};



template <class T> void Vector<T>::ascending_sort () {
    for(int i=0; i< size-1; i++)
        for(int j=i+1; j < size; j++)
            if(array[i] > array[j])
                swap(array[i], array[j]);
}
template <> void Vector<Mystring>::ascending_sort () {
    for(int i=0; i< size-1; i++)
        for(int j=i+1; j < size; j++)
            if(array[i].isGreater(array[j]))
                swap(array[i], array[j]);
}
```

```
template <> void Vector<char*>::ascending_sort () {
    for(int i=0; i< size-1; i++)
        for(int j=i+1; j < size; j++)
            if(strcmp(array[i], array[j]) > 0)
                swap(array[i], array[j]);
}

template <class T> void Vector<T>::swap (T& a, T& b) {
    T tmp = a;
    a = b;
    b = tmp;
}

template <class T> Vector<T>::VectIter::VectIter (Vector& x) {
    v = &x;
    index = 0;
}

template <class T> Vector<T>::Vector(int sz) {
    size=sz;
    array = new T [sz];
    assert (array != NULL);
}

template <class T> Vector<T>::~Vector() {
    delete [] array;
    array = NULL;
}

template <class T> T& Vector<T>::operator[] (int i) {
  return array[i];
}

template <class T> T Vector<T>::VectIter::operator* () {
    return v -> array[index];
}

template <class T> T Vector<T>::VectIter::operator++ () {
    // cout << "\n(size: " << v->size << ")";
    if (v->size == (index + 1)) {
        // cout << "resetting index\n";
        index = 0;
```

```
    } else {
        // cout << "index to " << index + 1 << "\n";
        index++;
    }
    return v->array[index];
}

template <class T> T Vector<T>::VectIter::operator++ (int) {
    // cout << "\n(size: " << v->size << ")";
    T ret = v->array[index];
    if (v->size == (index + 1)) {
        // cout << "resetting index\n";
        index = 0;
    } else {
        // cout << "index to " << index + 1 << "\n";
        index++;
    }
    return ret;
}

template <class T> T Vector<T>::VectIter::operator-- () {
    // cout << "\n(size: " << v->size << ")";
    if (index == 0) {
        index = v->size - 1;
        // cout << "maxing index to " << index << endl;
    } else {
        index--;
        // cout << "setting index to " << index << endl;
    }
    return v->array[index];
}

template <class T> T Vector<T>::VectIter::operator-- (int) {
    // cout << "\n(size: " << v->size << ")";
    T ret = v->array[index];
    if (index == 0) {
        index = v->size - 1;
        // cout << "maxing index to " << index << endl;
    } else {
        index--;
        // cout << "setting index to " << index << endl;
    }
```

```
    return ret;
}



int main() {
    Vector<int> x(3);
    x[0] = 999;
    x[1] = -77;
    x[2] = 88;
    Vector<int>::VectIter iter(x);
    cout << "\n\nThe first element of vector x contains: " << *iter;
    // the code between the  #if 0 and #endif is ignored by
    // compiler. If you change it to #if 1, it will be compiled
    #if 1
        cout << "\nTesting an <int> Vector: " << endl;;

        cout << "\n\nTesting sort";
        x.ascending_sort();

        for (int i=0; i<3 ; i++)
            cout << endl << iter++;

        cout << "\n\nTesting Prefix --:";
        for (int i=0; i<3 ; i++)
            cout << endl << --iter;

        cout << "\n\nTesting Prefix ++:";
        for (int i=0; i<3 ; i++)
            cout << endl << ++iter;

        cout << "\n\nTesting Postfix --";
        for (int i=0; i<3 ; i++)
            cout << endl << iter--;

        cout << endl;

        cout << "Testing a <String> Vector: " << endl;
        Vector<Mystring> y(3);
        y[0] = "Bar";
        y[1] = "Foo";
        y[2] = "All";;
```

```
        Vector<Mystring>::VectIter iters(y);

        cout << "\n\nTesting sort";
        y.ascending_sort();

        for (int i=0; i<3 ; i++)
            cout << endl << iters++;

        cout << "\n\nTesting Prefix --:";
        for (int i=0; i<3 ; i++)
            cout << endl << --iters;

        cout << "\n\nTesting Prefix ++:";
        for (int i=0; i<3 ; i++) {
            cout << endl << ++iters;
        }

        cout << "\n\nTesting Postfix --";
        for (int i=0; i<3 ; i++)
            cout << endl << iters--;

        cout << endl; cout << "Testing a <char *> Vector: " << endl;
        Vector<char*> z(3);
        z[0] = (char*)"Orange";
        z[1] = (char*)"Pear";
        z[2] = (char*)"Apple";

        Vector<char*>::VectIter iterchar(z);

        cout << "\n\nTesting sort";
        z.ascending_sort();

        for (int i=0; i<3 ; i++)
            cout << endl << iterchar++;

    #endif
        cout << "\nPrgram Terminated Successfully." << endl;
        return 0;
}
```

Terminal Output:

```
The first element of vector x contains: 999
Testing an <int> Vector:


Testing sort
-77
88
999

Testing Prefix --:
999
88
-77

Testing Prefix ++:
88
999
-77

Testing Postfix --
-77
999
88
Testing a <String> Vector:


Testing sort
All

Bar

Foo


Testing Prefix --:
Foo

Bar

All
```

```
Testing Prefix ++:
Bar

Foo

All


Testing Postfix --
All

Foo

Bar

Testing a <char *> Vector:


Testing sort
Apple
Orange
Pear
Prgram Terminated Successfully.
```

Exercise B

lookupTable.h

```
// LookupTable.h
// ENEL 409 - WINTER 2004
// Completed by: M. Moussavi

#ifndef LOOKUPTABLE_H
#define LOOKUPTABLE_H
#include <iostream>

using namespace std;



// class LookupTable: GENERAL CONCEPTS
//
//    key/datum pairs are ordered.  The first pair is the pair with
//    the lowest key, the second pair is the pair with the second
//    lowest key, and so on.  This implies that you must be able to
```

```
//    compare two keys with the < operator.
//
//    Each LookupTable has an embedded iterator class that allows users
//    of the class to traverse trhough the list and  have acess to each
//    node.

#include "customer.h"

//    In this version of the LookupTable a new struct type called Pair
//    is introduced which represents a key/data pair.

template <class V, class T> class LookupTable;

template <class V, class T> struct Pair {
    //constructor
    Pair (V keyA, T datumA) : key(keyA), datum(datumA) {}

    V key;
    T datum;
};

template <class V, class T> class LT_Node {
    friend class LookupTable<V, T>;
    friend class Iterator;
private:
    Pair<V, T> pairM;
    LT_Node<V, T> *nextM;

    // This ctor should be convenient in insert and copy operations.
    LT_Node (const Pair<V, T>& pairA, LT_Node<V, T> *nextA);
};

template <class V, class T> class LookupTable {
public:
    class Iterator {
        friend class LookupTable;
        LookupTable *LT;
        LT_Node<T, V>* cursor;

    public:
        Iterator () : LT(0) {}
        Iterator (LookupTable & x) : LT(&x) {}
```

```
        const T& operator* ();
        const T& operator++ ();
        const T& operator++ (int);
        int operator! ();

        void step_fwd () {
            assert(LT->cursor_ok());
            LT->step_fwd();
        }
    };

    LookupTable ();
    LookupTable (const LookupTable& source);
    LookupTable& operator= (const LookupTable& rhs);
    ~LookupTable();

    LookupTable& begin ();

    int size () const;
    // PROMISES: Returns number of keys in the table.

    int cursor_ok () const;
    // PROMISES:
    //   Returns 1 if the cursor is attached to a key/datum pair,
    //   and 0 if the cursor is in the off-list state.

    const V& cursor_key () const;
    // REQUIRES: cursor_ok()
    // PROMISES: Returns key of key/datum pair to which cursor is attached.

    const T& cursor_datum () const;
    // REQUIRES: cursor_ok()
    // PROMISES: Returns datum of key/datum pair to which cursor is attached.

    void insert (const Pair<V, T>& pariA);
    // PROMISES:
    //   If keyA matches a key in the table, the datum for that
    //   key is set equal to datumA.
    //   If keyA does not match an existing key, keyA and datumM are
    //   used to create a new key/datum pair in the table.
    //   In either case, the cursor goes to the off-list state.
```

```
    void remove (const V& keyA);
    // PROMISES:
    //   If keyA matches a key in the table, the corresponding
    //   key/datum pair is removed from the table.
    //   If keyA does not match an existing key, the table is unchanged.
    //   In either case, the cursor goes to the off-list state.

    void find (const V& keyA);
    // PROMISES:
    //   If keyA matches a key in the table, the cursor is attached
    //   to the corresponding key/datum pair.
    //   If keyA does not match an existing key, the cursor is put in
    //   the off-list state.

    void go_to_first ();
    // PROMISES: If size() > 0, cursor is moved to the first key/datum pair
    //   in the table.

    void step_fwd ();
    // REQUIRES: cursor_ok()
    // PROMISES:
    //   If cursor is at the last key/datum pair in the list, cursor
    //   goes to the off-list state.
    //   Otherwise the cursor moves forward from one pair to the next.

    void make_empty ();
    // PROMISES: size() == 0.



    friend  ostream& operator << <V, T> (ostream& os,const LookupTable<V, T>& lt);

private:
    int sizeM;
    LT_Node<V, T> *headM;
    LT_Node<V, T> *cursorM;

    void destroy();
    // Deallocate all nodes, set headM to zero.

    void copy(const LookupTable& source);
    // Establishes *this as a copy of source.  Cursor of *this will
    // point to the twin of whatever the source's cursor points to.
```

```
};

#endif

template <class V, class T> LookupTable<V, T>& LookupTable<V, T>::begin () {
    cursorM = headM;
    return *this;
}

template <class V, class T> LT_Node<V, T>::LT_Node (const Pair<V, T>& pairA, LT_Node<V, T> *nextA)
: pairM(pairA), nextM(nextA) {
}

template <class V, class T> LookupTable<V, T>::LookupTable ()
: sizeM(0), headM(0), cursorM(0) {
}

template <class V, class T> LookupTable<V, T>::LookupTable (const LookupTable<V, T>& source) {
    copy(source);
}

template <class V, class T> LookupTable<V, T>& LookupTable<V, T>::operator= (const LookupTable<V, T>& rhs)
{
    if (this != &rhs) {
        destroy();
        copy(rhs);
    }
    return *this;
}

template <class V, class T> LookupTable<V, T>::~LookupTable () {
    destroy();
}

template <class V, class T> int LookupTable<V, T>::size () const {
    return sizeM;
}

template <class V, class T> int LookupTable<V, T>::cursor_ok () const {
    return cursorM != 0;
}
```

```
template <class V, class T> const V& LookupTable<V, T>::cursor_key () const {
    assert(cursor_ok());
    return cursorM->pairM.key;
}

template <class V, class T> const T& LookupTable<V, T>::cursor_datum () const {
    assert(cursor_ok());
    return cursorM->pairM.datum;
}

template <class V, class T> void LookupTable<V, T>::insert (const Pair<V, T>& pairA) {
    // Add new node at head?
    if (headM == 0 || pairA.key < headM->pairM.key) {
        headM = new LT_Node<V, T>(pairA, headM);
        sizeM++;
    } else if (pairA.key == headM->pairM.key) {
        headM->pairM.datum = pairA.datum;
    } else {
        LT_Node<V, T>* before= headM;
        LT_Node<V, T>* after=headM->nextM;

        while(after!=NULL && (pairA.key > after->pairM.key)) {
            before=after;
            after=after->nextM;
        }

        if(after!=NULL && pairA.key == after->pairM.key) {
            after->pairM.datum = pairA.datum;
        } else {
            before->nextM = new LT_Node<V, T>(pairA, before->nextM);
            sizeM++;
        }
    }
}

template <class V, class T> void LookupTable<V, T>::remove (const V& keyA) {
    if (headM == 0 || keyA < headM->pairM.key)
        return;

    LT_Node<V, T>* doomed_node = 0;
    if (keyA == headM->pairM.key) {
        doomed_node = headM;
```

```
        headM = headM->nextM;
        sizeM--;
    } else {
        LT_Node<V, T> *before = headM;
        LT_Node<V, T> *maybe_doomed = headM->nextM;
        while(maybe_doomed != 0 && keyA > maybe_doomed->pairM.key) {
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->nextM;
        }

        if (maybe_doomed != 0 && maybe_doomed->pairM.key == keyA) {
            doomed_node = maybe_doomed;
            before->nextM = maybe_doomed->nextM;
            sizeM--;
        }
    }
    delete doomed_node;           // Does nothing if doomed_node == 0.
}

// This place-holder for find was added in order to
// allow successful linking when you're testing insert and remove.
// Replace it with a definition that works.

template <class V, class T> void LookupTable<V, T>::find (const V& keyA) {
    LT_Node<V, T> *ptr=headM;
    while (ptr!=NULL && ptr->pairM.key != keyA) {
        ptr=ptr->nextM;
    }
    cursorM = ptr;
}

template <class V, class T> void LookupTable<V, T>::go_to_first () {
    cursorM = headM;
}

template <class V, class T> void LookupTable<V, T>::step_fwd () {
    assert(cursor_ok());
    cursorM = cursorM->nextM;
}

template <class V, class T> void LookupTable<V, T>::make_empty () {
    destroy();
```

```
    sizeM = 0;
    cursorM = 0;
}

template <class V, class T> void LookupTable<V, T>::destroy () {
    LT_Node<V, T> *ptr = headM;
    while (ptr != NULL) {
        headM=headM->nextM;
        delete ptr;
        ptr=headM;
    }
    cursorM = NULL;
    sizeM=0;
}

template <class V, class T> void LookupTable<V, T>::copy(const LookupTable<V, T>& source) {
    headM=0;
    cursorM =0;

    if(source.headM ==0)
        return;



    for(LT_Node<V, T> *p = source.headM; p != 0; p=p->nextM) {
        insert(Pair<V, T>(p->pairM.key, p->pairM.datum));
        if(source.cursorM == p)
        find(p->pairM.key);
    }
}

template <class V, class T> ostream& operator << (ostream& os, const LookupTable<V, T>& lt) {
    if (lt.cursor_ok())
        os << lt.cursor_key() << "  " << lt.cursor_datum();
    else
        os<<"Not Found.";

    return os;
}

template <class V, class T> const T& LookupTable<V, T>::Iterator::operator* () {
    assert(LT ->cursor_ok());
    return LT->cursor_datum();
```

```
}

template <class V, class T> const T& LookupTable<V, T>::Iterator::operator++ () {
    assert(LT->cursor_ok());
    const T & x = LT->cursor_datum();
    LT->step_fwd();
    return x;
}

template <class V, class T> const T& LookupTable<V, T>::Iterator::operator++ (int) {
    assert(LT->cursor_ok());

    LT->step_fwd();
    return LT->cursor_datum();
}



template <class V, class T> int LookupTable<V, T>::Iterator::operator! () {
    return (LT->cursor_ok());
}
```

mainLab10ExB.cpp

```
// completed by: Mitchell Sawatzky

#include <assert.h>
#include <iostream>
#include "lookupTable.h"
#include "customer.h"
#include <cstring>
using namespace std;

template <class V, class T> void print(LookupTable<V, T>& lt);
template <class V, class T> void try_to_find(LookupTable<V, T>& lt, V key);

// void test_Customer();

//Uncomment the following function calls when ready to test template class LookupTable
void test_String();
void test_integer();

int main()
```

```
{

 //create and test a a lookup table of type <int, Customer>
 // test_Customer();
 // system("clear");

 // Uncomment the following function calls when ready to test template class LookupTable
 // create and test a a lookup table of type <int, String>
 test_String();
 system("clear");

 // Uncomment the following function calls when ready to test template class LookupTable
 // create and test a a lookup table of type <int, int>
 test_integer();

  cout<<"\n\nProgram terminated successfully.\n\n";

  return 0;
}



template <class V, class T> void print(LookupTable<V, T>& lt)
{
  if (lt.size() == 0)
    cout << "  Table is EMPTY.\n";
  for (lt.go_to_first(); lt.cursor_ok(); lt.step_fwd()) {
    cout << lt << endl;
  }
}



template <class V, class T> void try_to_find(LookupTable<V, T>& lt, V key)
{
  lt.find(key);
  if (lt.cursor_ok())
    cout << "\nFound key:"  << lt;
  else
    cout << "\nSorry, I couldn't find key: " << key << " in the table.\n";
}
```

```
/*void test_Customer()
  //creating a lookup table for customer objects.
  {
    cout<<"\nCreating and testing Customers Lookup Table-<no template>...\n";
    LookupTable<int, Customer> lt;

    // Insert using new keys.
    Customer a("Joe", "Morrison", "11 St. Calgary.", "(403)-1111-123333");
    Customer b("Jack", "Lewis", "12 St. Calgary.", "(403)-1111-123334");
    Customer c("Tim", "Hardy", "13 St. Calgary.", "(403)-1111-123335");
    lt.insert(Pair (8002, a));
    lt.insert(Pair (8004,c));
    lt.insert(Pair (8001,b));

    assert(lt.size() == 3);
    lt.remove(8004);
    assert(lt.size() == 2);
    cout << "\nPrinting table after inserting 3 new keys and 1 removal...\n";
    print(lt);

    // Pretend that a user is trying to look up customers info.

    cout << "\nLet's look up some names ...\n";
    try_to_find(lt, 8001);
    try_to_find(lt, 8000);

    // test Iterator
    cout << "\nTesing and using  iterator ...\n";
    LookupTable::Iterator it = lt.begin();
    cout <<"\nThe first node contains: " <<*it <<endl;

    while (!it) {
      cout <<++it << endl;
    }

    //test copying
    lt.go_to_first();
    lt.step_fwd();
    LookupTable clt(lt);
```

```
    assert(strcmp(clt.cursor_datum().getFname(),"Joe")==0);

    cout << "\nTest copying: keys should be 8001, and 8002\n";
    print(clt);
    lt.remove(8002);

    //Assignment operator check.
    clt= lt;

    cout << "\nTest assignment operator: key should be 8001\n";
    print(clt);

    //Wipe out the entries in the table.
    lt.make_empty();
    cout << "\nPrinting table for the last time: Table should be empty...\n";
    print(lt);



    cout << "***----Finished tests on Customers Lookup Table <not template>-----***\n";
    cout << "PRESS RETURN TO CONTINUE.";
    cin.get();

}*/


// Uncomment and modify the following funciton when ready to test LookupTable<int,Mystring>

void test_String()

  // creating lookuptable for Mystring objects
  {
    cout<<"\nCreating and testing LookupTable <int, Mystring> .....\n";
    LookupTable<int, Mystring> lt;

    // Insert using new keys.

    Mystring a("I am an ENEL-409 student.");
    Mystring b("C++ is a powerful language for engineers but it's not easy.");
    Mystring c ("Winter 2004");

    lt.insert(Pair<int, Mystring> (8002,a));
    lt.insert(Pair<int, Mystring> (8001,b));
```

```
    lt.insert(Pair<int, Mystring> (8004,c));

    //assert(lt.size() == 3);
    //lt.remove(8004);
    //assert(lt.size() == 2);
    cout << "\nPrinting table after inserting 3 new keys and  and 1 removal...\n";
    print(lt);

    // Pretend that a user is trying to look up customers info.

    cout << "\nLet's look up some names ...\n";
    try_to_find(lt, 8001);
    try_to_find(lt, 8000);
    // test Iterator
    LookupTable<int, Mystring>::Iterator it = lt.begin();
    cout <<"\nThe first node contains: " <<*it <<endl;

    while (!it) {
      cout <<++it << endl;
    }

    //test copying
    lt.go_to_first();
    lt.step_fwd();
    LookupTable<int, Mystring> clt(lt);
    assert(strcmp(clt.cursor_datum().c_str(),"I am an ENEL-409 student.")==0);

    cout << "\nTest copying: keys should be 8001, and 8002\n";
    print(clt);
    lt.remove(8002);

    //Assignment operator check.
    clt= lt;

    cout << "\nTest assignment operator: key should be 8001\n";
    print(clt);




    // Wipe out the entries in the table.
    lt.make_empty();
```

```
    cout << "\nPrinting table for the last time: Table should be empty ...\n";
    print(lt);

    cout << "***----Finished Lab 4 tests on <int> <Mystring>-----***\n";
    cout << "PRESS RETURN TO CONTINUE.";
    cin.get();
  }



// Uncomment and modify the following funciton when ready to test LookupTable<int,int>

void test_integer()

  //creating look table of integers

  {
    cout<<"\nCreating and testing LookupTable <int, int> .....\n";
    LookupTable<int, int>  lt;

    // Insert using new keys.
    lt.insert(Pair<int, int>(8002,9999));
    lt.insert(Pair<int, int>(8001,8888));
    lt.insert(Pair<int, int>(8004,8888));
    assert(lt.size() == 3);
    lt.remove(8004);
    assert(lt.size() == 2);
    cout << "\nPrinting table after inserting 3 new keys and  and 1 removal...\n";
    print(lt);

    // Pretend that a user is trying to look up customers info.

    cout << "\nLet's look up some names ...\n";
    try_to_find(lt, 8001);
    try_to_find(lt, 8000);

    // test Iterator
    LookupTable<int, int>::Iterator it = lt.begin();

    while (!it) {
      cout <<++it << endl;
```

```
    }

    //test copying
    lt.go_to_first();
    lt.step_fwd();
    LookupTable<int, int> clt(lt);
    assert(clt.cursor_datum()== 9999);

    cout << "\nTest copying: keys should be 8001, and 8002\n";
    print(clt);
    lt.remove(8002);

    //Assignment operator check.
    clt= lt;

    cout << "\nTest assignment operator: key should be 8001\n";
    print(clt);



    // Wipe out the entries in the table.
    lt.make_empty();
    cout << "\nPrinting table for the last time: Table should be empty ...\n";
    print(lt);

    cout << "***----Finished Lab 4 tests on <int> <int>-----***\n";

  }
```

Terminal Output

```
    Creating and testing LookupTable <int, Mystring> .....

    Printing table after inserting 3 new keys and  and 1 removal...
    C++ is a powerful language for engineers but it's not easy.
    I am an ENEL-409 student.

    Let's look up some names ...

    Found key:C++ is a powerful language for engineers but it's not easy.
    Sorry, I couldn't find key: 8000 in the table.

    The first node contains: 8001
```

```
8002

Test copying: keys should be 8001, and 8002
C++ is a powerful language for engineers but it's not easy.
I am an ENEL-409 student.

Test assignment operator: key should be 8001
C++ is a powerful language for engineers but it's not easy.

Printing table for the last time: Table should be empty ...
  Table is EMPTY.
***----Finished Lab 4 tests on <int> <Mystring>-----***
PRESS RETURN TO CONTINUE.

Creating and testing LookupTable <int, int> .....

Printing table after inserting 3 new keys and  and 1 removal...
8888
9999

Let's look up some names ...
Found key:8888
Sorry, I couldn't find key: 8000 in the table.
9999

Test copying: keys should be 8001, and 8002
8888
9999

Test assignment operator: key should be 8001
8888

Printing table for the last time: Table should be empty ...
  Table is EMPTY
***----Finished Lab 4 tests on <int> <int>-----***


Program terminated successfully.
```