Name: Alexa Astorino
Course Name: Principles of Software Design
Course Code: ENSF 480
Assignment Number: Lab 3
Submission Date and Time: 04/10/2018 12:00pm

Exercise A

// iterator.cpp

```
// ENSF 480 - Fall 2018 - Lab 3, Ex A
// M. Moussavi: Sept 26, 2018
// Alexa Astorino
#include <iostream>
#include <assert.h>
#include "mystring2.h"

using namespace std;


template <class T>
class Vector {
public:

  class VectIter{
    friend class Vector<T>;
  private:
    Vector<T> *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
public:
};

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]);
}

// specilaization for char*
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]);
}

//specilaization for Mystring
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])) == 1)
        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>
T Vector<T>::VectIter::operator *()
{
  return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator++() {
  index++;
  if (index >= v -> size) {
    index = 0;

  }
  return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator++(int i) {
  int temp = index;
  index++;
  if (index >= v -> size) {
    index = 0;

  }
  return v -> array[temp];
}

template <class T>
T Vector<T>::VectIter::operator--() {
  index--;
  if (index < 0) {
    index = (v -> size) - 1;


  }
  return v -> array[index];
}

template <class T>
T Vector<T>::VectIter::operator--(int i) {
  int temp = index;
  index--;
  if (index < 0) {
    index = (v -> size) - 1;

  }
  return v -> array[temp];
}
```

```
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];
}


ostream& operator << (ostream& os, const Mystring& s){
  return os << s.c_str();
}


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;
}
```

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

```
// ENSF 480 - Fall 2018 - Lab 3, Ex B
// M. Moussavi: Sept 26, 2018
// Alexa Astorino
#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"
//typedef int LT_Key; - class K now
//typedef Customer LT_Datum; - class U now

template <class K, class U>
struct Pair
{
  //constructor
  Pair(K keyA, U datumA):key(keyA), datum(datumA)
  {
  }
```

```
  K key;
  U datum;
};

template<class K, class U>
class LT_Node {
  template <class T, class D> friend class LookupTable;
private:
  Pair<K,U> pairM;
  LT_Node *nextM;

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

template <class K, class U>
class LookupTable {
 public:
  // Nested class
  class Iterator {
    friend class LookupTable<K,U>;
    LookupTable<K,U> *LT;

  public:
    Iterator():LT(0){}
    Iterator(LookupTable & x): LT(&x){}
    const U& operator *();
    const U& operator ++();
    const U& operator ++(int);
    int operator !();

    void step_fwd(){  assert(LT->cursor_ok());
    LT->step_fwd();}
  }; // End of class Iterator

  LookupTable();
  LookupTable(const LookupTable& source);
  LookupTable<K,U>& operator =(const LookupTable& rhs);
  ~LookupTable();

  LookupTable<K,U>& begin();

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

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

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

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

void insert(const Pair<K,U>& pairA);
// 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 cursorM goes to the off-list state.

void remove(const K& 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 cursorM goes to the off-list state.

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

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

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

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

template <typename O, typename S>
friend ostream& operator << (ostream& os, const LookupTable<O,S>& lt);


 private:
  int sizeM;
  LT_Node<K,U> *headM;
  LT_Node<K,U> *cursorM;

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

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

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

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

template <class K, class U>
LookupTable<K,U>::LookupTable()
  : sizeM(0), headM(0), cursorM(0)
{
}

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

template <class K, class U>
```

```
LookupTable<K,U>& LookupTable<K,U> ::operator =(const LookupTable& rhs)
{
  if (this != &rhs) {
    destroy();
    copy(rhs);
  }
  return *this;
}

template <class K, class U>
LookupTable<K,U>::~LookupTable()
{
  destroy();
}

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

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

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

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

template <class K, class U>
void LookupTable<K,U>::insert(const Pair<K,U>& pairA)
{
  // Add new node at head?
  if (headM == 0 || pairA.key < headM->pairM.key) {
```

```
    headM = new LT_Node<K,U>(pairA, headM);
    sizeM++;
  }

  // Overwrite datum at head?
  else if (pairA.key == headM->pairM.key)
    headM->pairM.datum = pairA.datum;

  // Have to search ...

  else {
    LT_Node<K,U>* before= headM;
    LT_Node<K,U>* 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<K,U>(pairA, before->nextM);
          sizeM++;
      }
  }
}

template <class K, class U>
void LookupTable<K,U>::remove(const K& keyA)
{

  if (headM == 0 || keyA < headM->pairM.key)
    return;

  LT_Node<K,U>* doomed_node = 0;
  if (keyA == headM->pairM.key) {
    doomed_node = headM;
    headM = headM->nextM;
    sizeM--;
  }
  else {
```

```
    LT_Node<K,U> *before = headM;
    LT_Node <K,U> *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.
}

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

    }
   cursorM = ptr;
}

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

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

template <class K, class U>
void LookupTable<K,U>::make_empty()
{
  destroy();
  sizeM = 0;
```

```
  cursorM = 0;
}

template <class K, class U>
void LookupTable<K,U>::destroy()
{

  LT_Node<K,U> *ptr = headM;
  while (ptr!=NULL)
    {
      headM=headM->nextM;
      delete ptr;
      ptr=headM;

    }
  cursorM = NULL;
  sizeM=0;
}

template <class K, class U>
void LookupTable<K,U>::copy(const LookupTable& source)
{

  headM=0;
  cursorM =0;

  if(source.headM ==0)
    return;

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

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

  return os;
```

```
}

template <class K, class U>
const U& LookupTable<K,U>::Iterator::operator *()
{
  assert(LT ->cursor_ok());
  return LT->cursor_datum();
}

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

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

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

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

// mainLab3ExB

```
// ENSF 480 - Fall 2018 - Lab 3, Ex B
// M. Moussavi: Sept 26, 2018
// Alexa Astorino
#include <assert.h>
#include <iostream>
#include "lookupTable.h"
#include "customer.h"
#include <cstring>
using namespace std;

template <class K, class U>
```

```
void print(LookupTable<K,U>& lt);

template <class K, class U>
void try_to_find(LookupTable<K,U>& lt, int key);

void test_Customer();

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

ostream& operator << (ostream& os, const Mystring& s){
  return os << s.c_str();
}


int main()
{
 //create and test a a lookup table of type <int, Customer>
 test_Customer();

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

 // Uncomment the following function calls when ready to test template class
LookupTable.
 // Then create and test a a lookup table of type <int, int>
 test_integer();
  cout<<"\n\nProgram terminated successfully.\n\n";
  return 0;
}

template <class K, class U>
void print(LookupTable<K,U>& 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 K, class U>
```

```
void try_to_find(LookupTable<K,U>& lt, int 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()
{
    cout<<"\nCreating and testing Customers Lookup Table <not 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<int, Customer> (8002,a));
    lt.insert(Pair<int, Customer> (8004,c));
    lt.insert(Pair<int, Customer> (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<int, Customer>::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, Customer> 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();

}


// When ready to test LookupTable<int , Mystrng> objects change #if 0 to #if 1

#if 1
void test_String()
{
    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();
}
#endif

// When ready to test LookupTable<int , int> objects change #if 0 to #if 1
```

```
#if 1
void test_integer()
{
    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";
}
#endif
```

output

```
Creating and testing Customers Lookup Table <not template>-...

Printing table after inserting 3 new keys and 1 removal...
8001  Nmae: Jack Lewis. Address: 12 St. Calgary. Phone: (403)-1111-123334
8002  Nmae: Joe Morrison. Address: 11 St. Calgary. Phone: (403)-1111-123333

Let's look up some names ...

Found key:8001  Nmae: Jack Lewis. Address: 12 St. Calgary. Phone:
(403)-1111-123334
Sorry, I couldn't find key: 8000 in the table.

Tesing and using  iterator ...

The first node contains: Nmae: Jack Lewis. Address: 12 St. Calgary. Phone:
(403)-1111-123334
Nmae: Jack Lewis. Address: 12 St. Calgary. Phone: (403)-1111-123334
Nmae: Joe Morrison. Address: 11 St. Calgary. Phone: (403)-1111-123333

Test copying: keys should be 8001, and 8002
8001  Nmae: Jack Lewis. Address: 12 St. Calgary. Phone: (403)-1111-123334
8002  Nmae: Joe Morrison. Address: 11 St. Calgary. Phone: (403)-1111-123333

Test assignment operator: key should be 8001
8001  Nmae: Jack Lewis. Address: 12 St. Calgary. Phone: (403)-1111-123334

Printing table for the last time: Table should be empty...
  Table is EMPTY.
***----Finished tests on Customers Lookup Table <not template>-----***
PRESS RETURN TO CONTINUE.

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

Printing table after inserting 3 new keys and  and 1 removal...
8001  C++ is a powerful language for engineers but it's not easy.
8002  I am an ENEL-409 student.

Let's look up some names ...

Found key:8001  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: C++ is a powerful language for engineers but it's
not easy.
C++ is a powerful language for engineers but it's not easy.
I am an ENEL-409 student.

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

Test assignment operator: key should be 8001
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...
8001  8888
8002  9999

Let's look up some names ...

Found key:8001  8888
Sorry, I couldn't find key: 8000 in the table.
8888
9999

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

Test assignment operator: key should be 8001
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.
```