ENSF 408
Lab 3
Shamin Rahman(30037908)
11 oct 2019
B01

EX A:

Output:

```
root@Rahman-Spectre:/mnt/c/Users/shami/Desktop/480/labs/lab3# ./a.out


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 <Mystring> Vector:


Testing sort
All
Bar
Foo

Testing Prefix --:
Foo
Bar
All
```

```
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.
```

Source code: (iterator.cpp)

```cpp
// iterator.cpp
// ENSF 480 - Fall 2019 - Lab 3, Ex A
// M. Moussavi: Sept 26, 2018
#include <iostream>
#include <assert.h>
#include <string>
#include <string.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<T> &x);

        T operator++(); // same as ++x
        //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); //same as x++
        // 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--(); //same as --X
        // 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();

    int &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 Vector<const char*>{
    private:
    int size;
```

```
    const char** array;

    public:
    Vector(int sz);
    ~Vector();
    void ascending_sort();
    void swap(const char*  &a, const char* &b);
    const char*& operator[](int i){
        return array[i];
    }

    // const char*& operator =(const char* rhs){

    // }
    class VectIter{
        friend class Vector<const char*>;
        private:
        Vector<const char*> *v; // points to a vector object of type T
        int index;

        public:
        VectIter(Vector<const char*> &x){
            v = &x;
            index = 0;
        }
        const char* operator++(){      //++x
            index++;
            if(index >= v->size)
                index = 0;
            return v->array[index];
        }
        const char* operator++(int){    //x++
            const char* res = v->array[index];
            index++;
            if(index >= v->size)
                index = 0;
            return res;
        }
        const char* operator--(){       //--x
            index--;
            if(index < 0)
                index = 0;
            return v->array[index];
        }
        const char* operator--(int){    //x--
            const char* res = v->array[index];
            index--;
            if(index < 0)
                index = 0;
            return res;
```

```
        }
        const char* operator*(){
            return v->array[index];
        }
    };
};

template<>
class Vector<Mystring>{
    private:
    int size;
    Mystring *array;

    public:
    Vector(int sz);
    ~Vector();
    friend ostream& operator << (ostream &out, const Mystring &c);

    void ascending_sort();
    void swap(Mystring  &a, Mystring &b);
    Mystring &operator[](int i){
        return array[i];
    }
    class VectIter{
        friend class Vector<Mystring>;
        private:
        Vector<Mystring> *v; // points to a vector object of type T
        int index;

        public:
        VectIter(Vector<Mystring> &x){
            v = &x;
            index = 0;
        }
        Mystring operator++(){      //++x
            index++;
            if(index >= v->size)
                index = 0;
            return v->array[index];
        }
        Mystring operator++(int){   //x++
            Mystring res = v->array[index];
            index++;
            if(index >= v->size)
                index = 0;
            return res;
        }
        Mystring operator--(){      //--x
            index--;
            if(index < 0)
```

```
                index = v->size - 1;
            return v->array[index];
        }
        Mystring operator--(int){   //x--
            Mystring res = v->array[index];
            index--;
            if(index < 0)
                index =  v->size - 1;
            return res;
        }
        Mystring operator*(){
            return v->array[index];
        }
    };
};

// ostream& operator << (ostream &out, const Mystring &c){
//  out << c.c_str();
//  return out;
// }

Vector<Mystring>::Vector(int sz){
    size = sz;
    array = new Mystring[size];
    assert(array != nullptr);
    //// strcpy(array, sz);
}
Vector<Mystring>::~Vector(){
    // for(int i = 0; i < size; i++){
    //  Mystring* yuh = array[0];
    //  delete [] yuh;
    // }
    delete [] array;
    array = NULL;
}
Vector<const char*>::Vector(int sz){
    size = sz;
    array = new const char*[size];
    assert(array != nullptr);
    //// strcpy(array, sz);
}
Vector<const char*>::~Vector(){
    // for(int i = 0; i < size; i++){
    //  delete [] array[i];
    // }
    delete [] array;
    array = NULL;
}

void Vector<const 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) //array[i] > array[j]
                swap(array[i], array[j]);
}
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]))   //array[i] > array[j]
                swap(array[i], array[j]);
}

template <class T>
T Vector<T>::VectIter::operator++()
{ // same as ++x
    index++;
    if (index >= v->size)
    {
        index = 0;
    }
    return v->array[index];
}

template <class T>
T Vector<T>::VectIter::operator++(int)
{ // same as x++
    int result = v->array[index];
    index++;
    if (index >= v->size)
    {
        index = 0;
    }
    return result;
}
template <class T>
T Vector<T>::VectIter::operator--()
{ //same as --X
    index--;
    if (index < 0)
    {
        index = v->size - 1;
    }
    return v->array[index];
}
template <class T>
T Vector<T>::VectIter::operator--(int)
{ //same as X--
    int result = v->array[index];
    index--;
    if (index < 0)
```

```
    {
        index = v->size - 1;
    }
    return result;
}
template <class T>
T Vector<T>::VectIter::operator*()
{
    return v->array[index];
}

template <class T> //TODO prob have to do specialization here
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 <class T>
void Vector<T>::swap(T &a, T &b)
{
    T tmp = a;
    a = b;
    b = tmp;
}

// template <>
void Vector<Mystring>::swap(Mystring &a, Mystring &b){
    Mystring tmp = a;
    a = b;
    b = tmp;
}
void Vector<const char*>::swap(const char* &a, const char* &b){
    const char* tmp = a;
    a = b;
    b = tmp;
}

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

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 <Mystring> 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 << endl << y;

    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<const char*> z(3);
    z[0] = "Orange";
    z[1] = "Pear";
    z[2] = "Apple";
    // creating copy of z called iterchar
    Vector<const 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;
}
```

EX B

```
root@Rahman-Spectre:/mnt/c/Users/shami/Desktop/480/labs/lab3# ./a.out

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.
8004  Winter 2004

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.
Winter 2004

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.
8004  Winter 2004

Test assignment operator: key should be 8001
8001  C++ is a powerful language for engineers but it's not easy.
8004  Winter 2004

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.
```

Source code: {lookuptable.h}

```
// LookupTable.h
// ENSF 480 - Fall 2019 - Lab 3, Ex A

#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"

template <class T, class K> class LookupTable;
template <class T, class K> struct Pair;
template <class T, class K> class LT_Node;

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

typedef int LT_Key;
typedef Customer LT_Datum;

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

    K key;
    T datum;
};

template <class T, class K>
class LT_Node{

    friend class LookupTable<T, K>;
private:
    Pair<T, K> pairM;
    LT_Node *nextM;
    // private:

    // This ctor should be convenient in insert and copy operations.
```

```
    LT_Node(const Pair<T, K> &pairA, LT_Node<T, K> *nextA);
};

template <class T, class K>
class LookupTable{
    private:
    int sizeM;
    LT_Node<T, K> *headM;
    LT_Node<T, K> *cursorM;

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

    void copy(const LookupTable<T, K> &source);
    // Establishes *this as a copy of source.  Cursor of *this will
    // point to the twin of whatever the source's cursor points to.
    public:
    class Iterator{
        friend class LookupTable;
        LookupTable<T, K> *LT;
        //    LT_Node* cursor;
        public:
        Iterator() : LT(0) {}
        Iterator(LookupTable<T, K> &x) : LT(&x) {}
        const T &operator*();
        const T &operator++();
        const T &operator++(int);
        int operator!();    //TODO dsas

        void step_fwd(){

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

    LookupTable();
    LookupTable(const LookupTable<T, K> &source);
    LookupTable<T, K> &operator=(const LookupTable<T, K> &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 K &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<T, K> &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 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 cursor goes to the off-list state.

void find(const K &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 << <K,D> (ostream& os,const LookupTable<K,D>& lt);

friend ostream &operator << (ostream &os, const LookupTable<T, K> &lt){
    if (lt.cursor_ok())
        os << lt.cursor_key() << "  " << lt.cursor_datum();
```

```
        else
            os << "Not Found.";

        return os;
    }
};

#endif

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

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

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

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

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

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

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

template <class T, class K>
```

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

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

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

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

```
}

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

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

    LT_Node<T, K> *doomed_node = 0;
    if (keyA == headM->pairM.key)
    {
        doomed_node = headM;
        headM = headM->nextM;
        sizeM--;
    }
    else
    {
        LT_Node<T, K> *before = headM;
        LT_Node<T, K> *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 T, class K>
void LookupTable<T, K>::find(const K &keyA){
    LT_Node<T, K> *ptr = headM;
    while (ptr != NULL && ptr->pairM.key != keyA)
    {
        ptr = ptr->nextM;
    }

    cursorM = ptr;
}

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

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

template <class T, class K>
void LookupTable<T, K>::make_empty(){
    destroy();
    sizeM = 0;
    cursorM = 0;
}

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

    LT_Node<T, K> *ptr = headM;
    while (ptr != NULL)
    {
        headM = headM->nextM;
        delete ptr;
        ptr = headM;
    }
    cursorM = NULL;
    sizeM = 0;
}

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

    headM = 0;
    cursorM = 0;

    if (source.headM == 0)
        return;

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

// template <class T, class K>
// ostream &operator << <T, K>(ostream &os, const LookupTable<T, K> &lt){
//  if (lt.cursor_ok())
//      os << lt.cursor_key() << "  " << lt.cursor_datum();
//  else
```

```
//      os << "Not Found.";

//  return os;
// }

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

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

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

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

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

Source code: {mainLab3ExB.h}

```
// ENSF 480 - Fall 2019 - Lab 3, Ex A
// M. Moussavi: Sept 26, 2019

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

template <class T, class K>
void print(LookupTable<T, K> &lt);

template <class T, class K>
void try_to_find(LookupTable<T, K> &lt, K key);
```

```
void test_Customer();
void test_String();
void  test_integer();

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

int main()
{

  //create and test a lookup table with an integer key value and Customer datum
  test_Customer();

  // 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();

  // 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 T, class K>
void print(LookupTable<T, K> &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 T, class K>
void try_to_find(LookupTable<T, K> &lt, K 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 <not template>-...\n";
LookupTable <Customer, int>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<Customer, int>(8002, a));
lt.insert(Pair<Customer, int>(8004, c));
lt.insert(Pair<Customer, int>(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<Customer, int>::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 <Customer, int>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 <Mystring, int>lt;// <Mystring, int>;

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

}
```