

Course Name:
Principles of Software Design

Lab Section:
B02

Course Code:
ENSF 480

Assignment Number:
Lab 1

Submission Date and Time:
20/09/2023

Completed By:
Findlay Brown: 30145677
David Rodriguez: 30145288

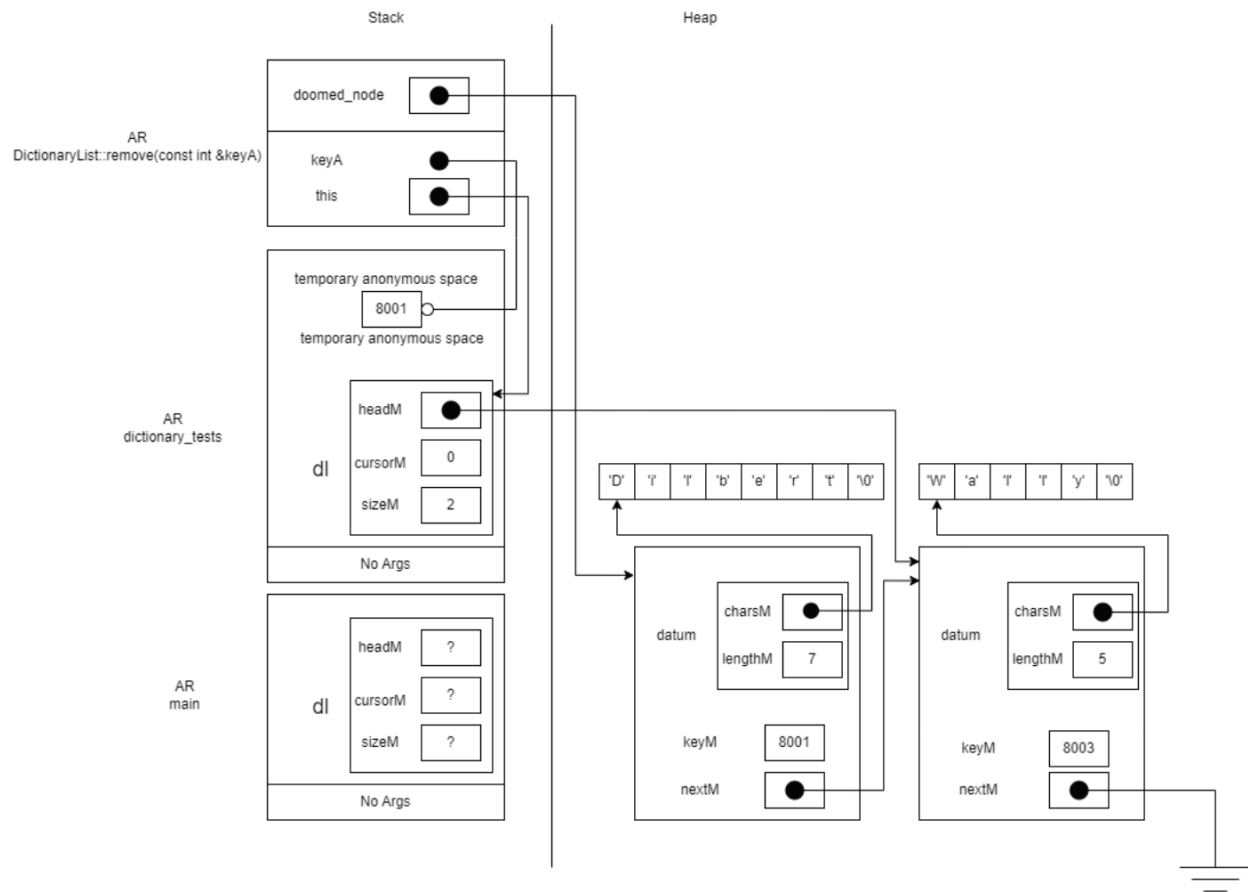
Exercise A

Program output and its order	Your explanation (why and where is the cause for this output)
constructor with int argument is called.	it is called at line 12 in exAmain. The statement, <code>Mystring c = 3</code> is interpreted by the compiler as a call to the constructor <code>Mystring::Mystring(int n)</code> .
default constructor is called. default constructor is called.	Both default constructors are called at line 18 in exAmain. The statement <code>Mystring x[2]</code> creates a 2 element array "x" and then calls the default constructor for each element in x.
constructor with char* argument is called.	Called at line 22 in exAmain. The statement, <code>Mystring *z = new Mystring("4");</code> is interpreted by the compiler as a call to the constructor <code>Mystring::Mystring(const char *s)</code> .
copy constructor is called. copy constructor is called.	Called at line 24 in exAmain. The copy constructor is called since objects are being copied into the append function as arguments.
destructor is called. destructor is called.	Called at line 24 in exAmain. The destructor is called since the arguments for the append function are going out of scope when the function finishes and as such the destructor is automatically called on.
copy constructor is called.	Called at line 26 in exAmain. The statement, <code>Mystring mars = x[0];</code> is interpreted by the compiler as a call to the copy constructor <code>Mystring::Mystring(const Mystring &source)</code> .
assignment operator called.	Called at line 28 in exAmain. The statement, <code>x[1] = x[0];</code> is interpreted by the compiler as a call to the assignment constructor <code>Mystring &Mystring::operator=(const Mystring &S)</code> .
constructor with char* argument is called. constructor with char* argument is called.	First call is on line 30 and the second on line 32 in exAmain. The statements, <code>Mystring jupiter("White");</code> and <code>ar[0] = new Mystring("Yellow");</code> are interpreted by the compiler as a call to the constructor <code>Mystring::Mystring(const char *s)</code> .

<p> destructor is called. destructor is called. destructor is called. destructor is called. destructor is called. </p>	<p>The first 4 destructors are called on line 34 after the closing bracket in exAmain and the last is called on line 37 in exAmain. The first 4 destructors are automatically called once the Mystring objects x[0], x[1], mars, and jupiter go out of scope. The last destructor on line 37 is called using the delete operator on ar[0].</p>
<p> constructor with char* argument is called. </p>	<p>Called on line 39 in exAmain. The statement, Mystring d = "Green"; is interpreted by the compiler as a call to the constructor Mystring::Mystring(const char *s).</p>
<p> Program terminated successfully. </p>	<p>Printed out on line 41 in exAmain. This is directly printed to the terminal since cout prints to the terminal.</p>
<p> destructor is called. destructor is called </p>	<p>Called at line 43 in exAmain. The program is finished as such that any object that wasn't deleted in the program has its destructor automatically called.</p>

Exercise B

Part 1:



Part 2

```
C:\Users\calga\OneDrive\Documents\GitHub\ENSF480\Lab01\ExerciseB>a.exe
```

```
Printing list just after its creation ...
```

```
List is EMPTY.
```

```
Printing list after inserting 3 new keys ...
```

```
8001 Dilbert
```

```
8002 Alice
```

```
8003 Wally
```

```
Printing list after removing two keys and inserting PointyHair ...
```

```
8003 Wally
```

```
8004 PointyHair
```

```
Printing list after changing data for one of the keys ...
```

```
8003 Sam
```

```
8004 PointyHair
```

```
Printing list after inserting 2 more keys ...
```

```
8001 Allen
```

```
8002 Peter
```

```
8003 Sam
```

```
8004 PointyHair
```

```
***----Finished dictionary tests-----***
```

```
Printing list--keys should be 315, 319
```

```
315 Shocks
```

```
319 Randomness
```

```
Printing list--keys should be 315, 319, 335
```

```
315 Shocks
```

```
319 Randomness
```

```
335 ParseErrors
```

```
Printing list--keys should be 315, 335
```

```
315 Shocks
```

```
335 ParseErrors
```

```
Printing list--keys should be 319, 335
```

```
319 Randomness
```

```
335 ParseErrors
```

```
Printing list--keys should be 315, 319, 335
```

```
315 Shocks
```

```
319 Randomness
```

```
335 ParseErrors
```

```
***----Finished tests of copying-----***
```

```
Let's look up some names ...
```

```
name for 8001 is: Allen.
```

```
Sorry, I couldn't find 8000 in the list.
```

```
name for 8002 is: Peter.
```

```
name for 8004 is: PointyHair.
```

```
***----Finished tests of finding -----***
```

Missing Functions

Find

```
void DictionaryList::find(const Key &keyA)
{
    Node *current = headM;
    while (current->nextM != 0 && current->keyM != keyA)
    {
        current = current->nextM;
    }
    if (current->keyM == keyA)
    {
        this->cursorM = current;
        return;
    }
    else
    {
        this->cursorM = 0;
    }
}
```

Destroy

```
void DictionaryList::destroy()
{
    this->cursorM = this->headM;
    this->sizeM = 0;
    while (this->cursorM != 0)
    {
        this->cursorM = this->headM->nextM;
        delete this->headM;
        this->headM = this->cursorM;
    }
}
```

Copy

```
void DictionaryList::copy(const DictionaryList &source)
{
    this->sizeM = source.sizeM;
    this->headM = new Node(source.headM->keyM, source.headM->datumM, nullptr);
    Node *tempSrcNode = source.headM;
    Node *tempNode = this->headM;
    while (tempSrcNode->nextM != 0)
    {

```

```

    tempSrcNode = tempSrcNode->nextM;
    tempNode->nextM = new Node(tempSrcNode->keyM, tempSrcNode->datumM, nullptr);
    tempNode = tempNode->nextM;
}
tempNode->nextM = nullptr;
if (source.cursorM == nullptr)
{
    this->cursorM = nullptr;
}
else
{
    this->find(source.cursorM->keyM);
}
}
}

```

Full code

exBmain.cpp

```

/*
 * File Name: exBmain.cpp
 * Assignment: Lab 1 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */

#include <assert.h>
#include <iostream>
#include "dictionaryList.h"

using namespace std;

DictionaryList dictionary_tests();

void test_copying();

void print(DictionaryList &dl);

void test_finding(DictionaryList &dl);

void test_operator_overloading(DictionaryList &dl);

int main()

```

```

{
    DictionaryList dl = dictionary_tests();

    test_copying();

    // Uncomment the call to test_copying when DictionaryList::copy is properly
defined
    test_finding(dl);
    // test_operator_overloading(dl);
    return 0;
}

```

```

DictionaryList dictionary_tests()

```

```

{
    DictionaryList dl;

    assert(dl.size() == 0);
    cout << "\nPrinting list just after its creation ...\n";
    print(dl);

    // Insert using new keys.
    dl.insert(8001, "Dilbert");
    dl.insert(8002, "Alice");
    dl.insert(8003, "Wally");
    assert(dl.size() == 3);
    cout << "\nPrinting list after inserting 3 new keys ...\n";
    print(dl);
    dl.remove(8002);
    dl.remove(8001);
    dl.insert(8004, "PointyHair");
    assert(dl.size() == 2);
    cout << "\nPrinting list after removing two keys and inserting PointyHair
...\n";
    print(dl);

    // Insert using existing key.
    dl.insert(8003, "Sam");
    assert(dl.size() == 2);
    cout << "\nPrinting list after changing data for one of the keys ...\n";
    print(dl);

    dl.insert(8001, "Allen");
    dl.insert(8002, "Peter");
}

```



```

    assert(dl.size() == 4);
    cout << "\nPrinting list after inserting 2 more keys ...\n";
    print(dl);

    cout << "***----Finished dictionary tests-----***\n\n";
    return dl;
}

void test_copying()
{
    DictionaryList one;

    // Copy an empty list.
    DictionaryList two;
    assert(two.size() == 0);

    // Copy a list with three entries and a valid cursor.
    one.insert(319, "Randomness");
    one.insert(315, "Shocks");
    one.insert(335, "ParseErrors");
    one.go_to_first();
    one.step_fwd();

    DictionaryList three(one);

    assert(three.cursor_datum().isEqual("Randomness"));
    one.remove(335);

    cout << "Printing list--keys should be 315, 319\n";
    print(one);

    cout << "Printing list--keys should be 315, 319, 335\n";
    print(three);

    // Assignment operator check.
    one = two = three = three;
    one.remove(319);
    two.remove(315);

    cout << "Printing list--keys should be 315, 335\n";
    print(one);

    cout << "Printing list--keys should be 319, 335\n";
    print(two);
}

```

```

    cout << "Printing list--keys should be 315, 319, 335\n";
    print(three);

    cout << "***-----Finished tests of copying-----***\n\n";
}

void print(DictionaryList &dl)
{
    if (dl.size() == 0)
        cout << "  List is EMPTY.\n";
    for (dl.go_to_first(); dl.cursor_ok(); dl.step_fwd())
    {
        cout << "    " << dl.cursor_key();
        cout << "    " << dl.cursor_datum().c_str() << '\n';
    }
}

void test_finding(DictionaryList &dl)
{
    // Pretend that a user is trying to look up names.
    cout << "\nLet's look up some names ...\n";

    dl.find(8001);
    if (dl.cursor_ok())
        cout << "  name for 8001 is: " << dl.cursor_datum().c_str() << ".\n";
    else
        cout << "  Sorry, I couldn't find 8001 in the list. \n";

    dl.find(8000);
    if (dl.cursor_ok())
        cout << "  name for 8000 is: " << dl.cursor_datum().c_str() << ".\n";
    else
        cout << "  Sorry, I couldn't find 8000 in the list. \n";

    dl.find(8002);
    if (dl.cursor_ok())
        cout << "  name for 8002 is: " << dl.cursor_datum().c_str() << ".\n";
    else
        cout << "  Sorry, I couldn't find 8002 in the list. \n";

    dl.find(8004);
    if (dl.cursor_ok())
        cout << "  name for 8004 is: " << dl.cursor_datum().c_str() << ".\n";
    else

```

```

        cout << "    Sorry, I couldn't find 8004 in the list. \n";

        cout << "***----Finished tests of finding -----***\n\n";
    }
#ifdef 0
void test_operator_overloading(DictionaryList& dl)
{
    DictionaryList dl2 = dl;
    dl.go_to_first();
    dl.step_fwd();
    dl2.go_to_first();

    cout << "\nTestig a few comparison and insertion operators." << endl;

    // Needs to overload >= and << (insertion operator) in class Mystring
    if(dl.cursor_datum() >= (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is greater than or equal " <<
dl2.cursor_datum();
    else
        cout << endl << dl2.cursor_datum() << " is greater than " <<
dl.cursor_datum();

    // Needs to overload <= for Mystring
    if(dl.cursor_datum() <= (dl2.cursor_datum()))
        cout << dl.cursor_datum() << " is less than or equal" <<
dl2.cursor_datum();
    else
        cout << endl << dl2.cursor_datum() << " is less than " <<
dl.cursor_datum();

    if(dl.cursor_datum() != (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is not equal to " <<
dl2.cursor_datum();
    else
        cout << endl << dl2.cursor_datum() << " is equal to " <<
dl.cursor_datum();

    if(dl.cursor_datum() > (dl2.cursor_datum()))
        cout << endl << dl.cursor_datum() << " is greater than " <<
dl2.cursor_datum();
    else
        cout << endl << dl.cursor_datum() << " is not greater than " <<
dl2.cursor_datum();

```

```

        if(dl.cursor_datum() < (dl2.cursor_datum()))
            cout << endl << dl.cursor_datum() << " is less than " <<
dl2.cursor_datum();
        else
            cout << endl << dl.cursor_datum() << " is not less than " <<
dl2.cursor_datum();
        if(dl.cursor_datum() == (dl2.cursor_datum()))
            cout << endl << dl.cursor_datum() << " is equal to " <<
dl2.cursor_datum();
        else
            cout << endl << dl.cursor_datum() << " is not equal to " <<
dl2.cursor_datum();
        cout << endl << "\nUsing square bracket [] to access elements of Mystring
objects. ";

        char c = dl.cursor_datum()[1];
        cout << endl << "The socond element of " << dl.cursor_datum() << " is: " <<
c;

        dl.cursor_datum()[1] = 'o';
        c = dl.cursor_datum()[1];
        cout << endl << "The socond element of " << dl.cursor_datum() << " is: " <<
c;

        cout << endl << "\nUsing << to display key/datum pairs in a Dictionary list:
\n";
        /* The following line is expected to display the content of the linked list
        * dl2 -- key/datum pairs. It should display:
        *   8001 Allen
        *   8002 Peter
        *   8003 Sam
        *   8004 PointyHair
        */
        cout << dl2;

        cout << endl << "\nUsing [] to display the datum only: \n";
        /* The following line is expected to display the content of the linked list
        * dl2 -- datum. It should display:
        *   Allen
        *   Peter
        *   Sam
        *   PointyHair
        */

```

```

    for(int i =0; i < dl2.size(); i++)
        cout << dl2[i] << endl;

    cout << endl << "\nUsing [] to display sequence of charaters in a datum: \n";
    /* The following line is expected to display the characters in the first node
    * of the dictionary. It should display:
    *   A
    *   l
    *   l
    *   e
    *   n
    */
    cout << dl2[0][0] << endl;
    cout << dl2[0][1] << endl;
    cout << dl2[0][2] << endl;
    cout << dl2[0][3] << endl;
    cout << dl2[0][4] << endl;

    cout << "\n\n***----Finished tests for overloading operators -----
***\n\n";
}
#endif

```

dictionaryList.cpp

```

/*
 * File Name: dictionaryList.cpp
 * Assignment: Lab 1 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */

#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "dictionaryList.h"
#include "mystring_B.h"

using namespace std;

Node::Node(const Key &keyA, const Datum &datumA, Node *nextA)
    : keyM(keyA), datumM(datumA), nextM(nextA)

```

```

{
}

DictionaryList::DictionaryList()
    : sizeM(0), headM(0), cursorM(0)
{
}

DictionaryList::DictionaryList(const DictionaryList &source)
{
    copy(source);
}

DictionaryList &DictionaryList::operator=(const DictionaryList &rhs)
{
    if (this != &rhs)
    {
        destroy();
        copy(rhs);
    }
    return *this;
}

DictionaryList::~DictionaryList()
{
    destroy();
}

int DictionaryList::size() const
{
    return sizeM;
}

int DictionaryList::cursor_ok() const
{
    return cursorM != 0;
}

const Key &DictionaryList::cursor_key() const
{
    assert(cursor_ok());
    return cursorM->keyM;
}

const Datum &DictionaryList::cursor_datum() const

```

```

{
    assert(cursor_ok());
    return cursorM->datumM;
}

void DictionaryList::insert(const int &keyA, const Mystring &datumA)
{
    // Add new node at head?
    if (headM == 0 || keyA < headM->keyM)
    {
        headM = new Node(keyA, datumA, headM);
        sizeM++;
    }

    // Overwrite datum at head?
    else if (keyA == headM->keyM)
        headM->datumM = datumA;

    // Have to search ...
    else
    {
        // POINT ONE

        // if key is found in list, just overwrite data;
        for (Node *p = headM; p != 0; p = p->nextM)
        {
            if (keyA == p->keyM)
            {
                p->datumM = datumA;
                return;
            }
        }

        // OK, find place to insert new node ...
        Node *p = headM->nextM;
        Node *prev = headM;

        while (p != 0 && keyA > p->keyM)
        {
            prev = p;
            p = p->nextM;
        }

        prev->nextM = new Node(keyA, datumA, p);
    }
}

```

```

        sizeM++;
    }
    cursorM = NULL;
}

void DictionaryList::remove(const int &keyA)
{
    if (headM == 0 || keyA < headM->keyM)
        return;

    Node *doomed_node = 0;

    if (keyA == headM->keyM)
    {
        doomed_node = headM;
        headM = headM->nextM;

        // POINT TWO
    }
    else
    {
        Node *before = headM;
        Node *maybe_doomed = headM->nextM;
        while (maybe_doomed != 0 && keyA > maybe_doomed->keyM)
        {
            before = maybe_doomed;
            maybe_doomed = maybe_doomed->nextM;
        }

        if (maybe_doomed != 0 && maybe_doomed->keyM == keyA)
        {
            doomed_node = maybe_doomed;
            before->nextM = maybe_doomed->nextM;
        }
    }
    if (doomed_node == cursorM)
        cursorM = 0;

    delete doomed_node; // Does nothing if doomed_node == 0.
    sizeM--;
}

void DictionaryList::go_to_first()
{
    cursorM = headM;
}

```



```

}

void DictionaryList::step_fwd()
{
    assert(cursor_ok());
    cursorM = cursorM->nextM;
}

void DictionaryList::make_empty()
{
    destroy();
    sizeM = 0;
    cursorM = 0;
}

// The following function are supposed to be completed by the stuents, as part
// of the exercise B part II. the given fucntion are in fact place-holders for
// find, destroy and copy, in order to allow successful linking when you're
// testing insert and remove. Replace them with the definitions that work.

void DictionaryList::find(const Key &keyA)
{
    Node *current = headM;
    while (current->nextM != 0 && current->keyM != keyA)
    {
        current = current->nextM;
    }
    if (current->keyM == keyA)
    {
        this->cursorM = current;
        return;
    }
    else
    {
        this->cursorM = 0;
    }
}

void DictionaryList::destroy()
{
    this->cursorM = this->headM;
    this->sizeM = 0;
    while (this->cursorM != 0)
    {
        this->cursorM = this->headM->nextM;
    }
}

```

```

        delete this->headM;
        this->headM = this->cursorM;
    }
}

void DictionaryList::copy(const DictionaryList &source)
{
    this->sizeM = source.sizeM;
    this->headM = new Node(source.headM->keyM, source.headM->datumM, NULL);
    Node *tempSrcNode = source.headM;
    Node *tempNode = this->headM;
    while (tempSrcNode->nextM != 0)
    {
        tempSrcNode = tempSrcNode->nextM;
        tempNode->nextM = new Node(tempSrcNode->keyM, tempSrcNode->datumM, NULL);
        tempNode = tempNode->nextM;
    }
    tempNode->nextM = nullptr;
    if (source.cursorM == nullptr)
    {
        this->cursorM = nullptr;
    }
    else
    {
        this->find(source.cursorM->keyM);
    }
}

```

dictionaryList.h

```

/*
 * File Name: dictionaryList.h
 * Assignment: Lab 1 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */

#ifndef DICTIONARY_H
#define DICTIONARY_H
#include <iostream>
using namespace std;

// class DictionaryList: 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 DictionaryList object has a "cursor" that is either attached
//  to a particular key/datum pair or is in an "off-list" state, not
//  attached to any key/datum pair.  If a DictionaryList is empty, the
//  cursor is automatically in the "off-list" state.

#include "mystring_B.h"

// Edit these typedefs to change the key or datum types, if necessary.
typedef int Key;
typedef Mystring Datum;

// THE NODE TYPE
//  In this exercise the node type is a class, that has a ctor.
//  Data members of Node are private, and class DictionaryList
//  is declared as a friend. For details on the friend keyword refer to your
//  lecture notes.

class Node
{
    friend class DictionaryList;

private:
    Key keyM;
    Datum datumM;
    Node *nextM;

    // This ctor should be convenient in insert and copy operations.
    Node(const Key &keyA, const Datum &datumA, Node *nextA);
};

class DictionaryList
{
public:
    DictionaryList();
    DictionaryList(const DictionaryList &source);
    DictionaryList &operator=(const DictionaryList &rhs);
    ~DictionaryList();

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

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

void insert(const Key &keyA, const Datum &datumA);
// 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 Key &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 Key &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.

private:
    int sizeM;
    Node *headM;
    Node *cursorM;

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

    void copy(const DictionaryList &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

```

Mystring_B.cpp

```

/*
 * File Name: mystring_B.cpp
 * Assignment: Lab 1 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */

#include "mystring_B.h"
#include <string.h>
#include <iostream>
using namespace std;

Mystring::Mystring()
{
    charsM = new char[1];

    // make sure memory is allocated.
    memory_check(charsM);
    charsM[0] = '\0';
    lengthM = 0;
}

```

```

Mystring::Mystring(const char *s)
    : lengthM(strlen(s))
{
    charsM = new char[lengthM + 1];

    // make sure memory is allocated.
    memory_check(charsM);

    strcpy(charsM, s);
}

Mystring::Mystring(int n)
    : lengthM(0), charsM(new char[n])
{
    // make sure memory is allocated.
    memory_check(charsM);
    charsM[0] = '\0';
}

Mystring::Mystring(const Mystring &source) : lengthM(source.lengthM), charsM(new
char[source.lengthM + 1])
{
    memory_check(charsM);
    strcpy(charsM, source.charsM);
}

Mystring::~Mystring()
{
    delete[] charsM;
}

int Mystring::length() const
{
    return lengthM;
}

char Mystring::get_char(int pos) const
{
    if (pos < 0 && pos >= length())
    {
        cerr << "\nERROR: get_char: the position is out of boundary.";
    }

    return charsM[pos];
}

```

```

const char *Mystring::c_str() const
{
    return charsM;
}

void Mystring::set_char(int pos, char c)
{
    if (pos < 0 && pos >= length())
    {
        cerr << "\nset_char: the position is out of boundary."
              << " Nothing was changed.";
        return;
    }

    if (c != '\0')
    {
        cerr << "\nset_char: char c is empty."
              << " Nothing was changed.";
        return;
    }

    charsM[pos] = c;
}

```

```

Mystring &Mystring::operator=(const Mystring &S)
{
    if (this == &S)
        return *this;
    delete[] charsM;
    lengthM = (int)strlen(S.charsM);
    charsM = new char[lengthM + 1];
    memory_check(charsM);
    strcpy(charsM, S.charsM);

    return *this;
}

```

```

Mystring &Mystring::append(const Mystring &other)
{
    char *tmp = new char[lengthM + other.lengthM + 1];
    memory_check(tmp);
    lengthM += other.lengthM;
    strcpy(tmp, charsM);
    strcat(tmp, other.charsM);
}

```

```

        delete[] charsM;
        charsM = tmp;

        return *this;
    }

    void Mystring::set_str(char *s)
    {
        delete[] charsM;
        lengthM = (int)strlen(s);
        charsM = new char[lengthM + 1];
        memory_check(charsM);

        strcpy(charsM, s);
    }

    int Mystring::isNotEqual(const Mystring &s) const
    {
        return (strcmp(charsM, s.charsM) != 0);
    }

    int Mystring::isEqual(const Mystring &s) const
    {
        return (strcmp(charsM, s.charsM) == 0);
    }

    int Mystring::isGreaterThan(const Mystring &s) const
    {
        return (strcmp(charsM, s.charsM) > 0);
    }

    int Mystring::isLessThan(const Mystring &s) const
    {
        return (strcmp(charsM, s.charsM) < 0);
    }

    void Mystring::memory_check(char *s)
    {
        if (s == 0)
        {
            cerr << "Memory not available.";
            exit(1);
        }
    }
}

```


mystring_B.h

```
/*
 * File Name: mystring_B.h
 * Assignment: Lab 1 Exercise B
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */

#include <iostream>
#include <string>
using namespace std;

#ifndef MYSTRING_H
#define MYSTRING_H

class Mystring
{
public:
    Mystring();
    // PROMISES: Empty string object is created.

    Mystring(int n);
    // PROMISES: Creates an empty string with a total capacity of n.
    //           In other words, dynamically allocates n elements for
    //           charsM, sets the lengthM to zero, and fills the first
    //           element of charsM with '\0'.

    Mystring(const char *s);
    // REQUIRES: s points to first char of a built-in string.
    // REQUIRES: Mystring object is created by copying chars from s.

    ~Mystring(); // destructor

    Mystring(const Mystring &source); // copy constructor

    Mystring &operator=(const Mystring &rhs); // assignment operator
    // REQUIRES: rhs is reference to a Mystring as a source
    // PROMISES: to make this-object (object that this is pointing to, as a copy
    //           of rhs.

    int length() const;
    // PROMISES: Return value is number of chars in charsM.
```

```

char get_char(int pos) const;
// REQUIRES: pos >= 0 && pos < length()
// PROMISES:
// Return value is char at position pos.
// (The first char in the charsM is at position 0.)

const char *c_str() const;
// PROMISES:
// Return value points to first char in built-in string
// containing the chars of the string object.

void set_char(int pos, char c);
// REQUIRES: pos >= 0 && pos < length(), c != '\0'
// PROMISES: Character at position pos is set equal to c.

Mystring &append(const Mystring &other);

// PROMISES: extends the size of charsM to allow concatenate other.charsM to
// to the end of charsM. For example if charsM points to "ABC", and
// other.charsM points to XYZ, extends charsM to "ABCXYZ".
//

void set_str(char *s);
// REQUIRES: s is a valid C++ string of characters (a built-in string)
// PROMISES: copies s into charsM, if the length of s is less than or equal
lengthM.
// Otherwise, extends the size of the charsM to s.lengthM+1, and
copies
// s into the charsM.

int isGreaterThan(const Mystring &s) const;
// REQUIRES: s refers to an object of class Mystring
// PROMISES: retruns true if charsM is greater than s.charsM.

int isLessThan(const Mystring &s) const;
// REQUIRES: s refers to an object of class Mystring
// PROMISES: retruns true if charsM is less than s.charsM.

int isEqual(const Mystring &s) const;
// REQUIRES: s refers to an object of class Mystring
// PROMISES: retruns true if charsM equal s.charsM.

int isNotEqual(const Mystring &s) const;
// REQUIRES: s refers to an object of class Mystring
// PROMISES: retruns true if charsM is not equal s.charsM.

```

```
private:
    int lengthM; // the string length - number of characters excluding \0
    char *charsM; // a pointer to the beginning of an array of characters,
    allocated dynamically.
    void memory_check(char *s);
    // PROMISES: if s points to NULL terminates the program.
};
#endif
```

Exercise C

company.h

```
/*
 * File Name: company.h
 * Assignment: Lab 1 Exercise C
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */
#ifndef COMPANY
#define COMPANY

#include <string>
#include <vector>

#include "employee.h"
#include "customer.h"
#include "address.h"
#include "date.h"
using namespace std;

class Company
{
private:
    string companyName;

    Address companyAddress;

    Date dateEstablished;

    vector<Employee> employees;
```

```

    vector<Customer> customers;

    // public: Functions
};

#endif

```

employee.h

```

/*
 * File Name: employee.h
 * Assignment: Lab 1 Exercise C
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */
#ifndef EMPLOYEE
#define EMPLOYEE

#include <string>
#include <vector>
#include "date.h"
#include "address.h"
using namespace std;

class Employee
{
private:
    enum State
    {
        active = 0,
        suspended,
        retired,
        fired
    };
    string name;
    Address address;
    Date dateOfBirth;
    State employmentStatus;

    // public: Functions
};

#endif

```

customer.h

```
/*
 * File Name: customer.h
 * Assignment: Lab 1 Exercise C
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */
#ifndef CUSTOMER
#define CUSTOMER

#include <string>
#include "address.h"
using namespace std;

class Customer
{
private:
    string name;

    Address address;

    string phoneNumber;

    // public: Functions
};

#endif
```

address.h

```
/*
 * File Name: address.h
 * Assignment: Lab 1 Exercise C
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */
#ifndef ADDRESS
#define ADDRESS

#include <string>
using namespace std;

class Address
```

```

{

private:
    string street;
    string postalCode;
    string city;
    string country;

    // public: Functions
};

#endif

```

date.h

```

/*
 * File Name: date.h
 * Assignment: Lab 1 Exercise C
 * Lab Section: B02
 * Completed by: Findlay Brown, David Rodriguez
 * Submission Date: Sept 20, 2023
 */
#ifndef DATE
#define DATE

class Date
{
private:
    int day;
    int month;
    int year;

    // public: Functions
};

#endif

```

Exercise D

See zip folder attached to submission.