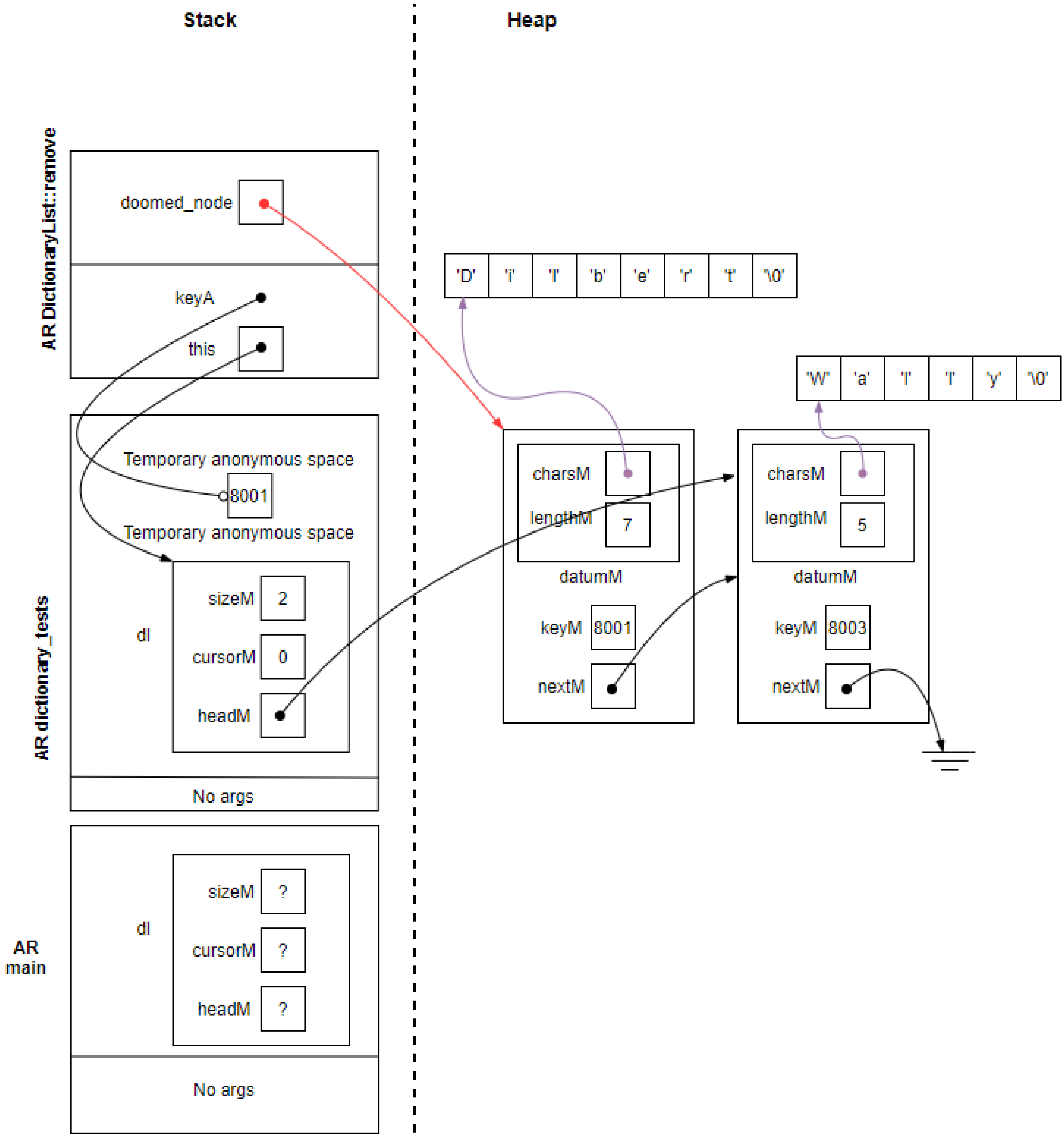
# Lab 6 Report

**Course: ENSF 619** - Fall 2020
**Lab #:** Lab 6
**Student Name:** Davis Allan, 10016543
**Submission Date:** Oct 30 2020

# Exercise A:

## AR Diagram:



## Exercise B:

### Source Code:

#### Class myString:

```
 /*
* File Name: myString.h
* Lab # and Assignment #: Lab 6 Exercise A and B
* Lab section: B01
* Completed by: Davis Allan, 10016543
* Submission Date: Oct 30 2020
*/
#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.

  bool operator >=(const Mystring& rhs) const; //overloading >= operator
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: returns true if this-object is greater than or equal to
(lexicographically)
  //            to rhs, false otherwise

  bool operator <=(const Mystring& rhs) const; //overloading <= operator
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: returns true if this-object is less than or equal to
(lexicographically)
  //            to rhs, false otherwise

  bool operator !=(const Mystring& rhs) const;
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: returns true if this-object not equal to
(lexicographically)
  //            to rhs, false otherwise

  bool operator >(const Mystring& rhs) const;
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: returns true if this-object is greater than
(lexicographically)
  //            to rhs, false otherwise

  bool operator <(const Mystring& rhs) const;
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: returns true if this-object is less than
(lexicographically)
  //            to rhs, false otherwise

  bool operator ==(const Mystring& rhs) const;
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: returns true if this-object is less equal to
(lexicographically)
  //            to rhs, false otherwise

  char& operator [](int i) const;
  //  REQUIRES: i <= lengthM
  //  PROMISES: returns the character at index i of charsM
```

```cpp
  friend ostream& operator<<(ostream& os, const Mystring& rhs);
//overloading << operator
  //  REQUIRES: rhs is a reference to a Mystring as a source
  //  PROMISES: outputs the string to the console

  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:copys s into charsM, if the length of s is less than or
equal lengthM.
  //          Othrewise, extends the size of the charsM to s.lengthM+1,
and copies
  //          s into the 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
```

```
 /*
* File Name: myString.cpp
* Lab # and Assignment #: Lab 6 Exercise A and B
* Lab section: B01
* Completed by: Davis Allan, 10016543
* Submission Date: Oct 30 2020
*/
#include "mystring.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;
}

bool Mystring::operator>=(const Mystring& rhs) const {
  return strcmp(this->charsM, rhs.charsM) >= 0;
}

bool Mystring::operator<=(const Mystring& rhs) const {
  return strcmp(this->charsM, rhs.charsM) <= 0;
}

bool Mystring::operator!=(const Mystring& rhs) const {
  return strcmp(this->charsM, rhs.charsM) != 0;
}

bool Mystring::operator>(const Mystring& rhs) const {
  return strcmp(this->charsM, rhs.charsM) > 0;
}

bool Mystring::operator<(const Mystring& rhs) const {
  return strcmp(this->charsM, rhs.charsM) < 0;
}

bool Mystring::operator==(const Mystring& rhs) const {
  return strcmp(this->charsM, rhs.charsM) == 0;
}

char& Mystring::operator[](int i) const {
  if (i > this->lengthM) {
    cout << "Index out of bounds! Exiting program" << endl;
    exit(1);
  }
  return this->charsM[i];
}

ostream& operator<<(ostream& os, const Mystring& rhs) {
  os << rhs.charsM;
  return os;
}

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);
}


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

### Class dictionaryList

```
 /*
* File Name: dictionaryList.h
* Lab # and Assignment #: Lab 6 Exercise A and B
* Lab section: B01
* Completed by: Davis Allan, 10016543
* Submission Date: Oct 30 2020
*/
#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.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.

  friend ostream& operator<<(ostream& os, DictionaryList dl);
  //  REQUIRES: dl is a reference to a DictionaryList object as a source
  //  PROMISES: outputs all of the Nodes in the DictionaryList to the
console

  const Mystring& operator[](int i);
  //  REQUIRES: i <= sizeM
  //  PROMISES: returns a reference to the datum at index i of the
DictionaryList

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
```

```
 /*
* File Name: dictionaryList.cpp
* Lab # and Assignment #: Lab 6 Exercise A and B
* Lab section: B01
* Completed by: Davis Allan, 10016543
* Submission Date: Oct 30 2020
*/
#include <assert.h>
#include <iostream>
#include <stdlib.h>
#include "dictionaryList.h"
#include "mystring.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;
}

ostream& operator<<(ostream& os, DictionaryList dl) {
  dl.go_to_first();

  while (dl.cursor_ok()) {
    os << dl.cursor_key() << " " << dl.cursor_datum() << endl;
    dl.step_fwd();
  }
  return os;
}

const Mystring& DictionaryList::operator[](int i) {
  if (i > sizeM) {
    cout << "Index out of bounds! Exiting program" << endl;
    exit(1);
  }

  int j = 0;
  go_to_first();
  while(j != i) {
    step_fwd();
    j++;
  }

  return cursor_datum();
}

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;
}



void DictionaryList::copy(const DictionaryList& source)
{
    if (source.headM == 0) {
        headM = 0;
        return;
    }

    headM = new Node (source.headM->keyM, source.headM->datumM, NULL);
    Node *newest_node = headM;

    const Node *source_node = source.headM;

    while (true) {
        source_node = source_node->nextM;
        if (source_node == 0)
            break;
        newest_node->nextM = new Node(source_node->keyM, source_node-
>datumM, NULL);
        newest_node = newest_node->nextM;
    }

    cursorM = source.cursorM;
    sizeM = source.sizeM;
}

void DictionaryList::find(const int& keyA)
{
    for (Node *p = headM; p != 0; p=p->nextM)
```

```
        if (keyA == p->keyM)
        {
            cout << "'" << keyA <<"' was found with datum value " << p->datumM.c_str() << ".\n";
            cursorM = p;
            return;
        }
    cout << "'" << keyA <<"' was not found.\n";
    cursorM = 0;
}

void DictionaryList::destroy()
{
    Node *p = headM;
    Node *prev;
    while (p != 0)
    {
        prev = p;
        p = p->nextM;
        delete prev;
    }
    headM = 0;
    sizeM = 0;
}
```

### exAMain code

```
 /*
* File Name: exAmain.cpp
* Lab # and Assignment #: Lab 6 Exercise A and B
* Lab section: B01
* Completed by: Davis Allan, 10016543
* Submission Date: Oct 30 2020
*/
#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_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_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() << endl;

    // Needs to overload <= for Mystring
    if(dl.cursor_datum() <= (dl2.cursor_datum()))
        cout << endl << 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 second element of "  << dl.cursor_datum() << "
is: " << c;

    dl.cursor_datum()[1] = 'o';
    c = dl.cursor_datum()[1];
    cout << endl << "The second 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";
}

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';
    }
}
```

### Program Output

```
PS C:\Users\davis\Desktop\ENSF 619\Labs\Lab6> .\test.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---------------------------***



Testig a few comparison and insertion operators.

Peter is greater than or equal Allen
Allen is less than Peter
Peter is not equal to Allen
Peter is greater than Allen
Peter is not less than Allen
Peter is not equal to Allen

Using square bracket [] to access elements of Mystring objects.
The second element of Peter is: e
The second element of Poter is: o

Using << to display key/datum pairs in a Dictionary list:
8001 Allen
```

```
8002 Peter
8003 Sam
8004 PointyHair


Using [] to display the datum only:
Allen
Peter
Sam
PointyHair


Using [] to display sequence of charaters in a datum:
A
l
l
e
n


***----Finished tests for overloading operators ----------***
```

## Exercise C and D:

### Source Code

#### Class MyVector

```
 /*
  * File Name: MyVector.java
  * Lab # and Assignment #: Lab 6 Exercise C and D
  * Lab section: B01
  * Completed by: Davis Allan, 10016543
  * Submission Date: Oct 30 2020
  */

 import java.util.ArrayList;

public class MyVector {

    private ArrayList<Item> storageM;
    private Sorter sortStrategy;

    public MyVector(int n) {
        storageM = new ArrayList<>(n);
    }

    public MyVector(ArrayList<Item> arr) {
        storageM = new ArrayList<>(arr.size());
        for (Item i : arr) {
            storageM.add(new Item(i.getItem()));
        }
    }

    public void add(Item value) {
        storageM.add(value);
    }

    public void setSortStrategy(Sorter s) {
        sortStrategy = s;
    }

    public void performSort() {
        sortStrategy.sort(storageM);
    }
    public void display() {
        for (Item i : storageM) {
            System.out.print(i.getItem() + " ");
        }
    }
}
```

### Class Sorter

```
 /*
  * File Name: Sorter.java
  * Lab # and Assignment #: Lab 6 Exercise C and D
  * Lab section: B01
  * Completed by: Davis Allan, 10016543
  * Submission Date: Oct 30 2020
  */

import java.util.ArrayList;

public interface Sorter {
    void sort(ArrayList<Item> arr);
}
```

### Class BubbleSorter

```
 /*
  * File Name: BubbleSorter.java
  * Lab # and Assignment #: Lab 6 Exercise C and D
  * Lab section: B01
  * Completed by: Davis Allan, 10016543
  * Submission Date: Oct 30 2020
  */

import java.util.ArrayList;

public class BubbleSorter implements Sorter {

    @Override
    //code adapted from https://www.geeksforgeeks.org/bubble-sort/
    public void sort(ArrayList<Item> arr) {
        int n = arr.size();
        for (int i = 0; i < n - 1; i++)
            for (int j = 0; j < n - i - 1; j++)
                if (arr.get(j).getItem() > arr.get(j + 1).getItem()) {

                    Item temp = arr.get(j);
                    arr.set(j, arr.get(j + 1));
                    arr.set(j + 1, temp);
                }
    }
}
```

### Class InsertionSorter

```
 /*
  * File Name: InsertionSorter.java
  * Lab # and Assignment #: Lab 6 Exercise C and D
  * Lab section: B01
  * Completed by: Davis Allan, 10016543
  * Submission Date: Oct 30 2020
  */
import java.util.ArrayList;

public class InsertionSorter implements Sorter {

    @Override
    //code adapted from https://www.geeksforgeeks.org/insertion-sort/
    public void sort(ArrayList<Item> arr) {
        int n = arr.size();

        for (int i = 1; i < n; i++) {
            Item key = arr.get(i);
            int j = i - 1;

            while (j >= 0 && arr.get(j).getItem() > key.getItem()) {
                arr.set(j + 1, arr.get(j));
                j = j - 1;
            }
            arr.set(j + 1, key);
        }
    }
}
```

### Class SelectionSorter

```java
 /*
  * File Name: SelectionSorter.java
  * Lab # and Assignment #: Lab 6 Exercise C and D
  * Lab section: B01
  * Completed by: Davis Allan, 10016543
  * Submission Date: Oct 30 2020
  */
import java.util.ArrayList;

public class SelectionSorter implements Sorter {

    @Override
    //code adapted from https://www.programiz.com/dsa/selection-sort
    public void sort(ArrayList<Item> arr) {
        int size = arr.size();

        for (int step = 0; step < size - 1; step++) {
            int min_idx = step;

            for (int i = step + 1; i < size; i++) {

                if (arr.get(i).getItem() < arr.get(min_idx).getItem()) {
                    min_idx = i;
                }
            }

            Item temp = arr.get(step);
            arr.set(step, arr.get(min_idx));
            arr.set(min_idx, temp);
        }
    }
}
```

### Class Item

```java
/*
 * File Name: Item.java
 * Lab # and Assignment #: Lab 6 Exercise C and D
 * Lab section: B01
 * Completed by: Davis Allan, 10016543
 * Submission Date: Oct 30 2020
 */
class Item {
    private Double item;
    public Item(Double value) {
        item = value;
    }

    public void setItem(Double value){
        item = value;
    }

    public Double getItem(){
        return item;
    }
}
```

### Class DemoStrategyPattern

```
 /*
  * File Name: DemoStrategyPattern.java
  * Lab # and Assignment #: Lab 6 Exercise C and D
  * Lab section: B01
  * Completed by: Davis Allan, 10016543
  * Submission Date: Oct 30 2020
  */
import java.util.ArrayList;
import java.util.Random;
public class DemoStrategyPattern {
    public static void main(String[] args) {
        // Create an object of MyVector<Double> with capacity of 50
elements
        MyVector v1 = new MyVector (50);

        // Create a Random object to generate values between 0
        Random rand = new Random();

        // adding 5 randomly generated numbers into MyVector object v1
       for(int i = 4; i >=0; i--) {
           Item item;
           item = new Item (Double.valueOf(rand.nextDouble()*100));
           v1.add(item);
       }

        // displaying original data in MyVector v1
       System.out.println("The original values in v1 object are:");
        v1.display();

        // choose algorithm bubble sort as a strategy to sort object v1
        v1.setSortStrategy(new BubbleSorter ());

        // perform algorithm bubble sort to v1
        v1.performSort();

        System.out.println("\nThe values in MyVector object v1 after
performing BoubleSorter is:");
        v1.display();

        // create a MyVector<Integer> object V2
        MyVector v2 = new MyVector (50);
        // populate v2 with 5 randomly generated numbers
        for(int i = 4; i >=0; i--) {
            Item item;
            item = new Item (Double.valueOf(rand.nextInt(50)));
            v2.add(item);
            }

        System.out.println("\nThe original values in v2 object are:");
        v2.display();
        v2.setSortStrategy(new InsertionSorter());;
        v2.performSort();
        System.out.println("\nThe values in MyVector object v2 after
performing InsertionSorter is:");
        v2.display();

        // create a MyVector<Integer> object V3

        MyVector v3 = new MyVector(10);

        for (int i = 0; i < 6; i++) {
            v3.add(new Item(Double.valueOf(rand.nextInt(75))));
        }

        v3.setSortStrategy(new SelectionSorter());
        System.out.println("\nThe original values in v3 object are:");
        v3.display();
        v3.performSort();
        System.out.println("\nThe values in MyVector object v3 after
performing SelectionSorter is:");
        v3.display();

        ArrayList<Item> arrayList = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            arrayList.add(new Item(Double.valueOf(rand.nextInt(75))));
        }

        System.out.println("\n\nTesting copy constructor of MyVector");
        MyVector v4 = new MyVector(arrayList);

        arrayList.remove(0);
        arrayList.remove(3);
        arrayList.get(0).setItem(-99.0);
        System.out.println("Printing arrayList after manipulating");
        for (Item i : arrayList) {
            System.out.print(i.getItem() + " ");
        }
        System.out.println("\nPrinting v4, expecting 5 elements.");
        v4.display();
    }
}
```

## Program Output

```
The original values in v1 object are:
64.03146172315903 12.79983292562379 48.01321860257013 15.886760460814076
50.09236992229936
The values in MyVector object v1 after performing BoubleSorter is:
12.79983292562379 15.886760460814076 48.01321860257013 50.09236992229936
64.03146172315903
The original values in v2 object are:
43.0 24.0 47.0 42.0 1.0
The values in MyVector object v2 after performing InsertionSorter is:
1.0 24.0 42.0 43.0 47.0
The original values in v3 object are:
33.0 66.0 67.0 5.0 57.0 54.0
The values in MyVector object v3 after performing SelectionSorter is:
5.0 33.0 54.0 57.0 66.0 67.0

Testing copy constructor of MyVector
Printing arrayList after manipulating
-99.0 63.0 64.0
Printing v4, expecting 5 elements.
68.0 43.0 63.0 64.0 73.0
Process finished with exit code 0
```