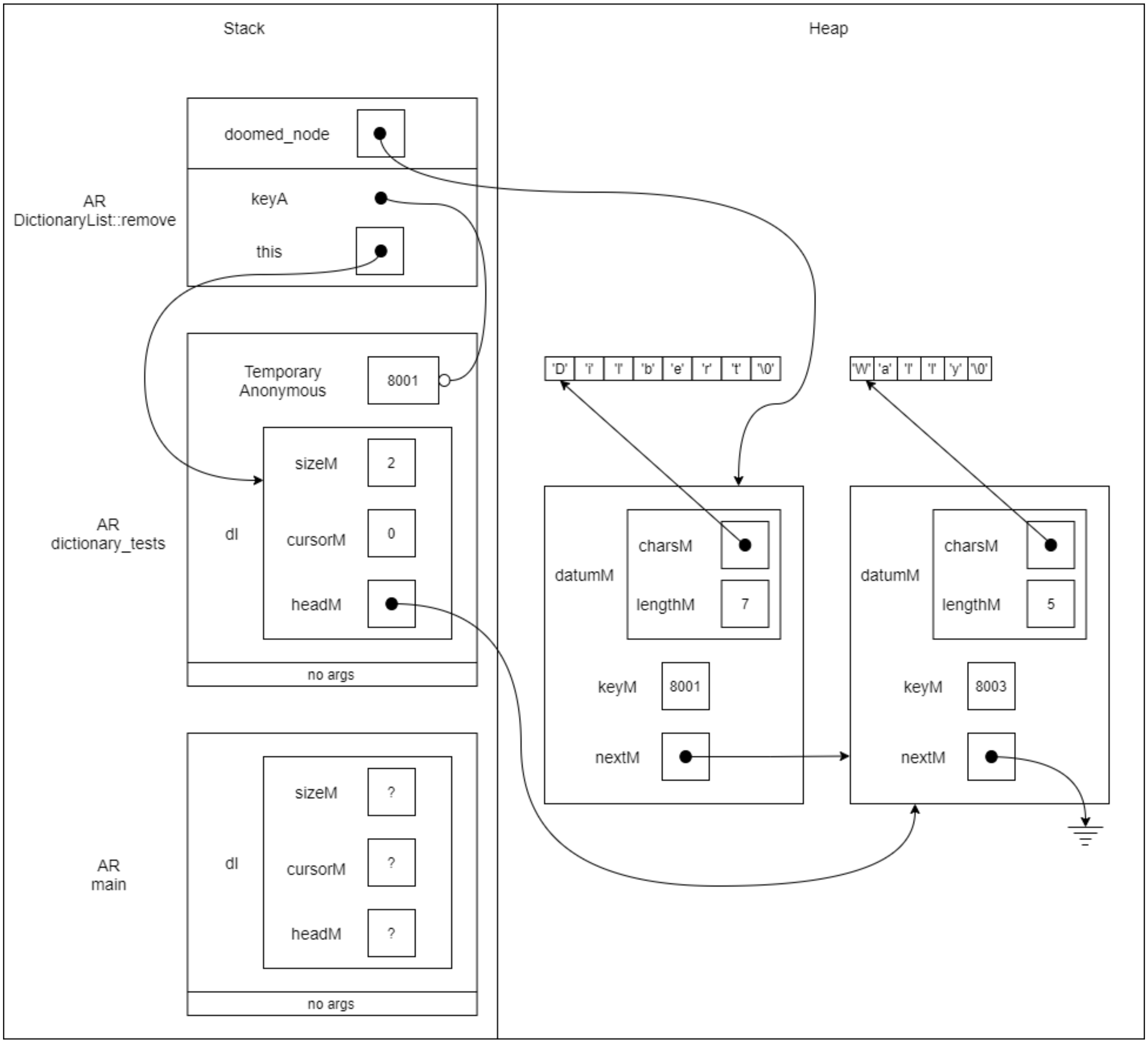
ENSF 614 – Fall 2021

Lab 6 – Tuesday, November 16

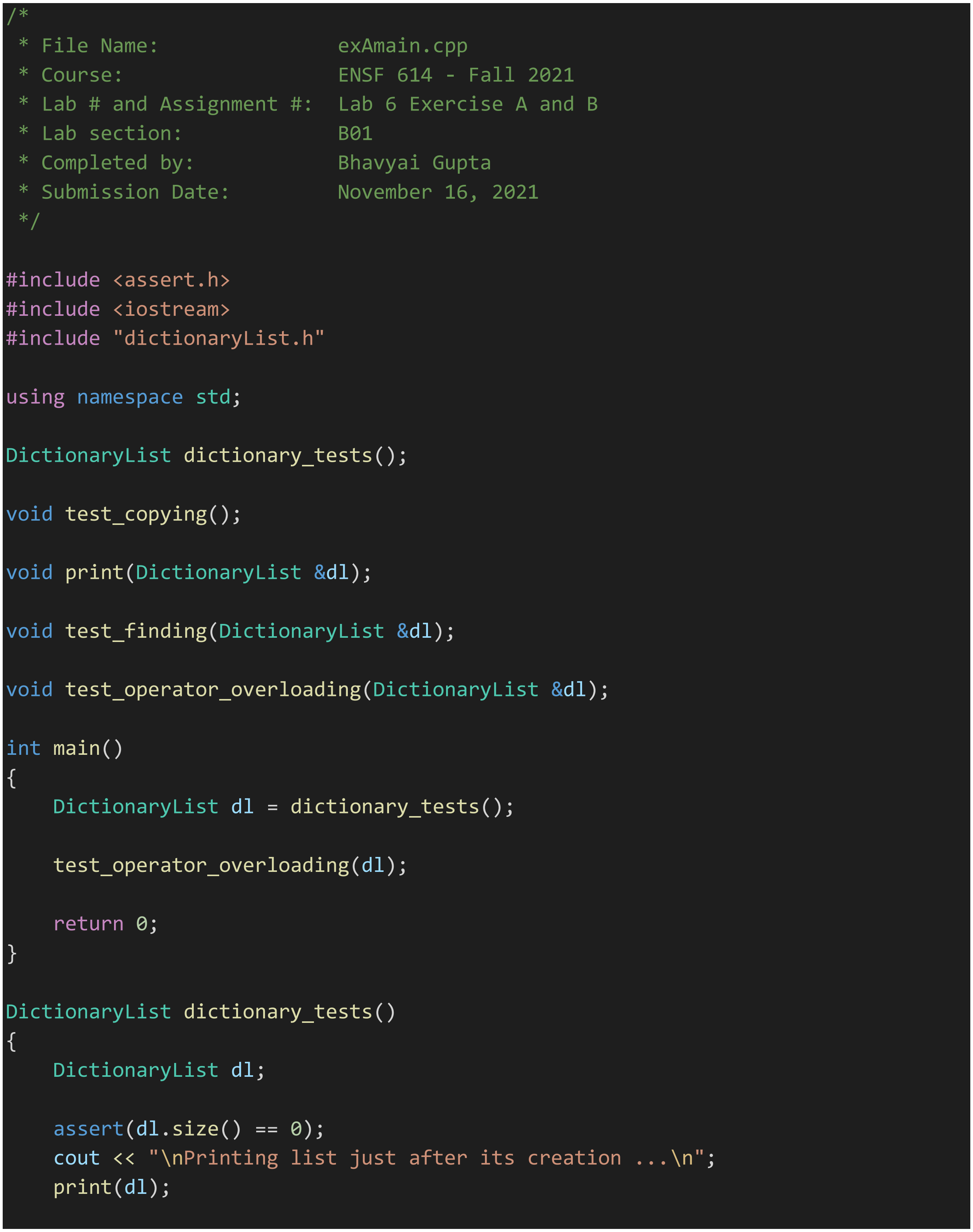
Student Name: Bhavyai Gupta

Submission date: November 16, 2021

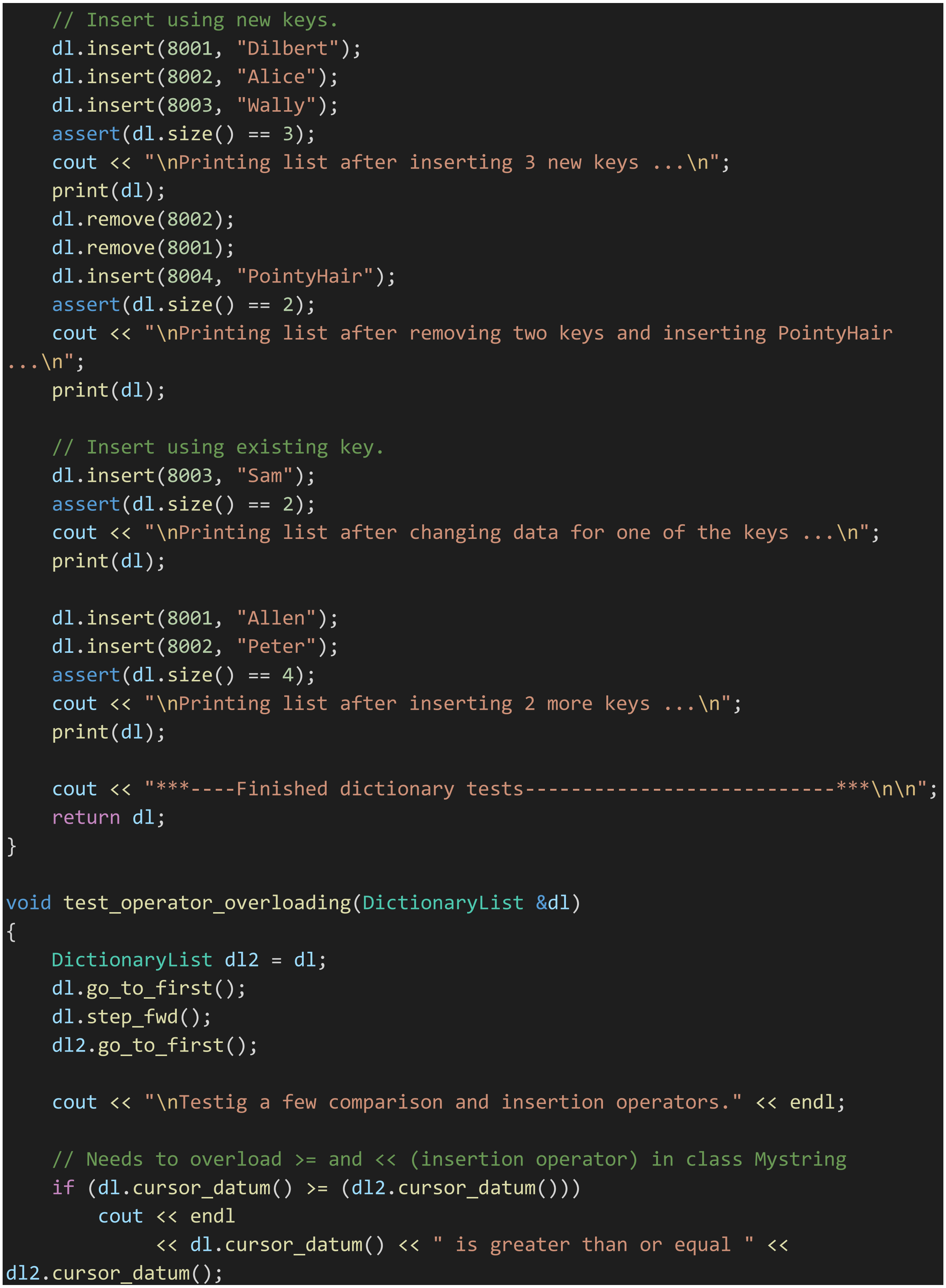
Exercise A – AR Diagram for Point Two



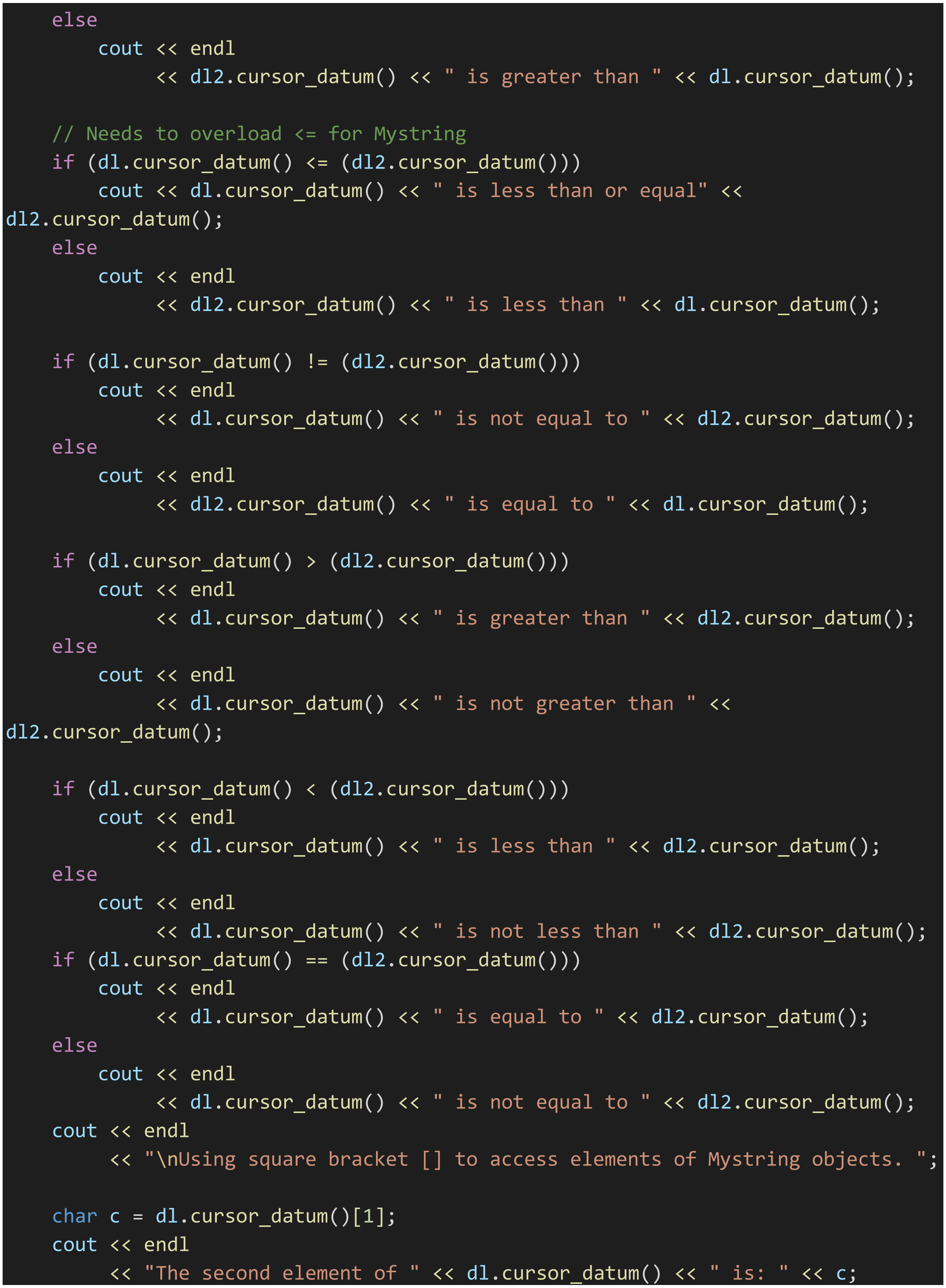
Exercise B – Source file exAmain.cpp



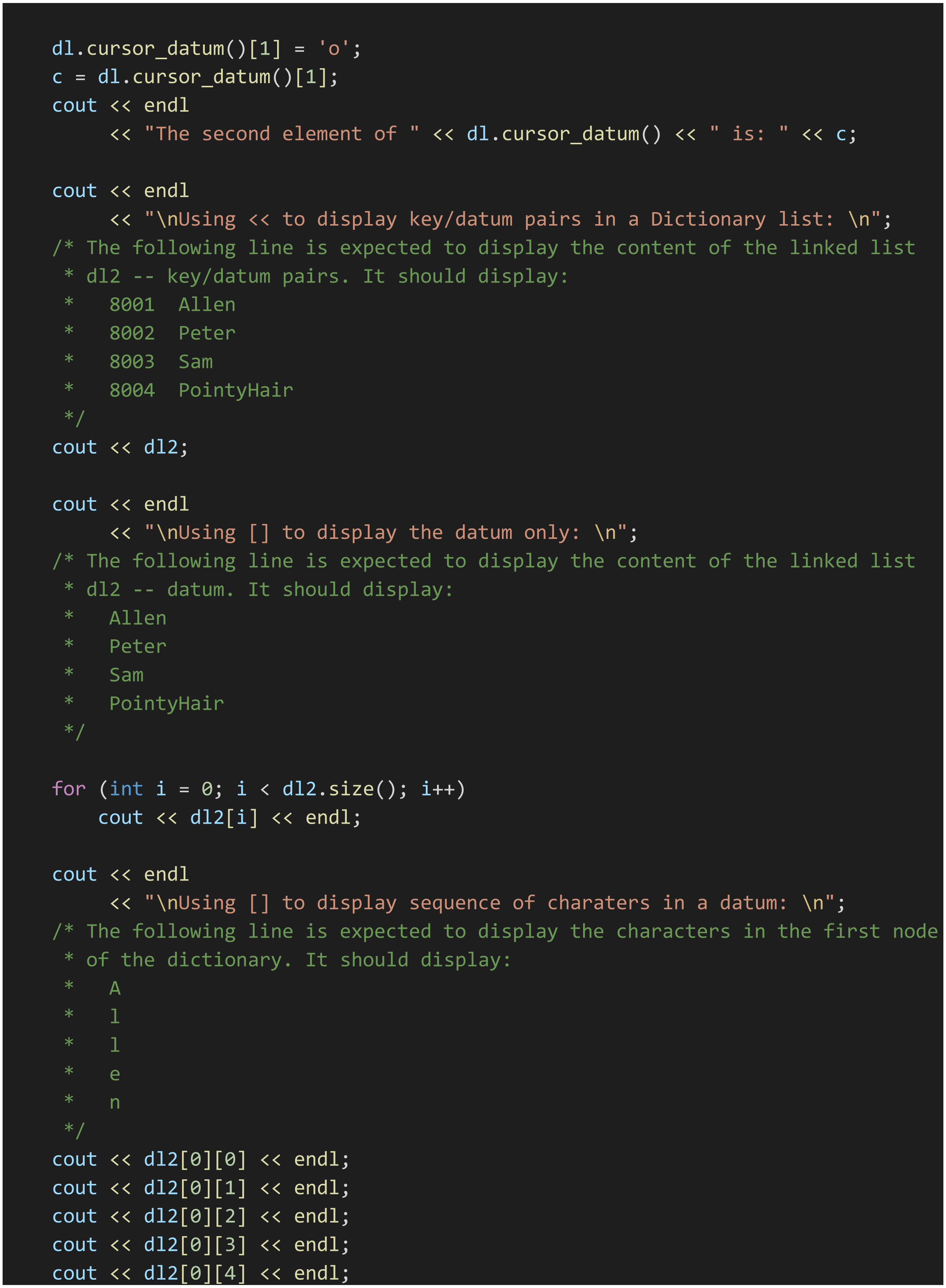
```
/*
 * File Name:                 exAmain.cpp
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise A and B
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

#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();

    // 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 second element of " << dl.cursor_datum() << " is: " << c;
```

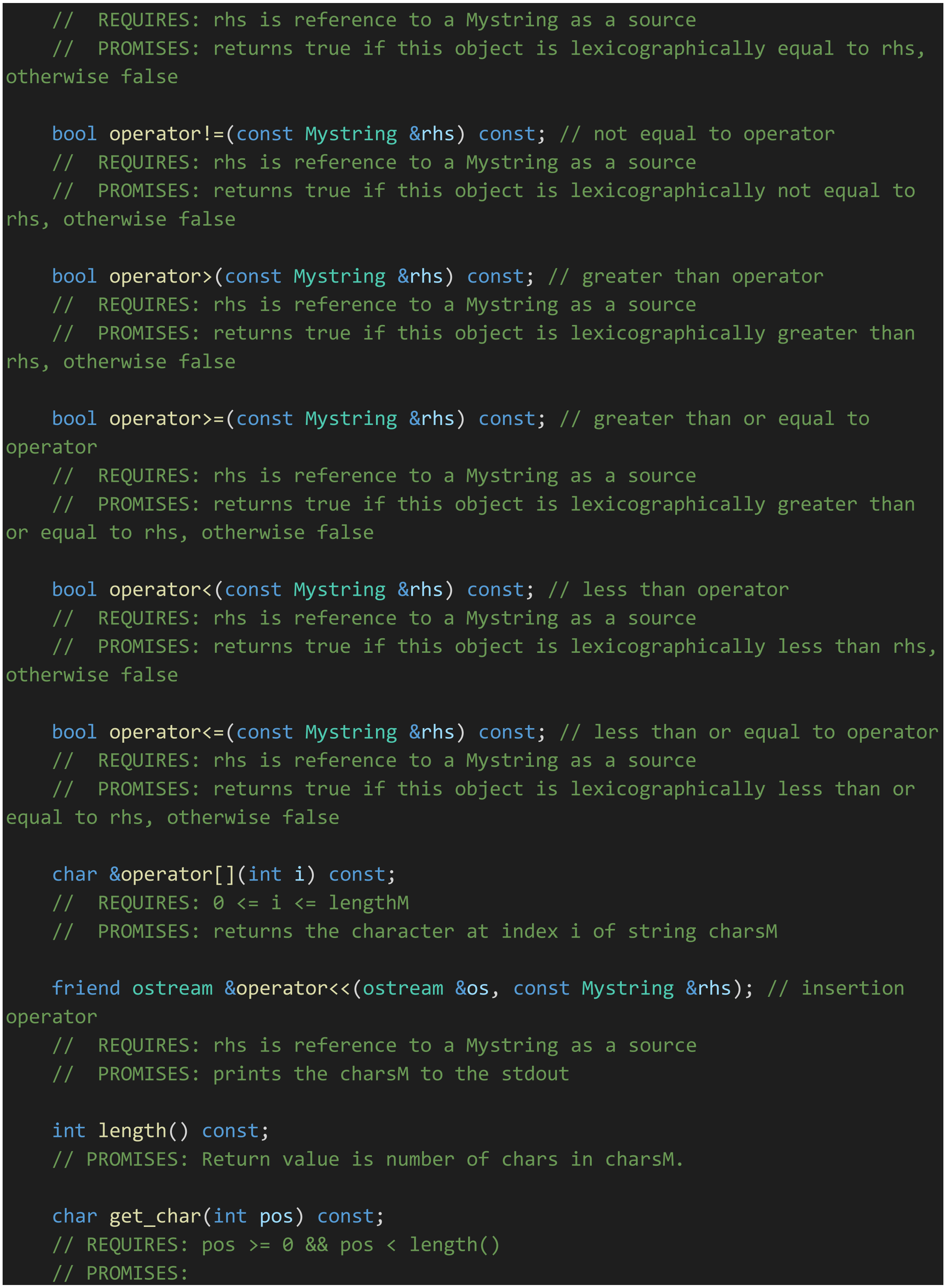


```
dl.cursor_datum()[1] = 'o';
c = dl.cursor_datum()[1];
cout << endl
     << "The second element of " << dl.cursor_datum() << " is: " << c;

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;
```

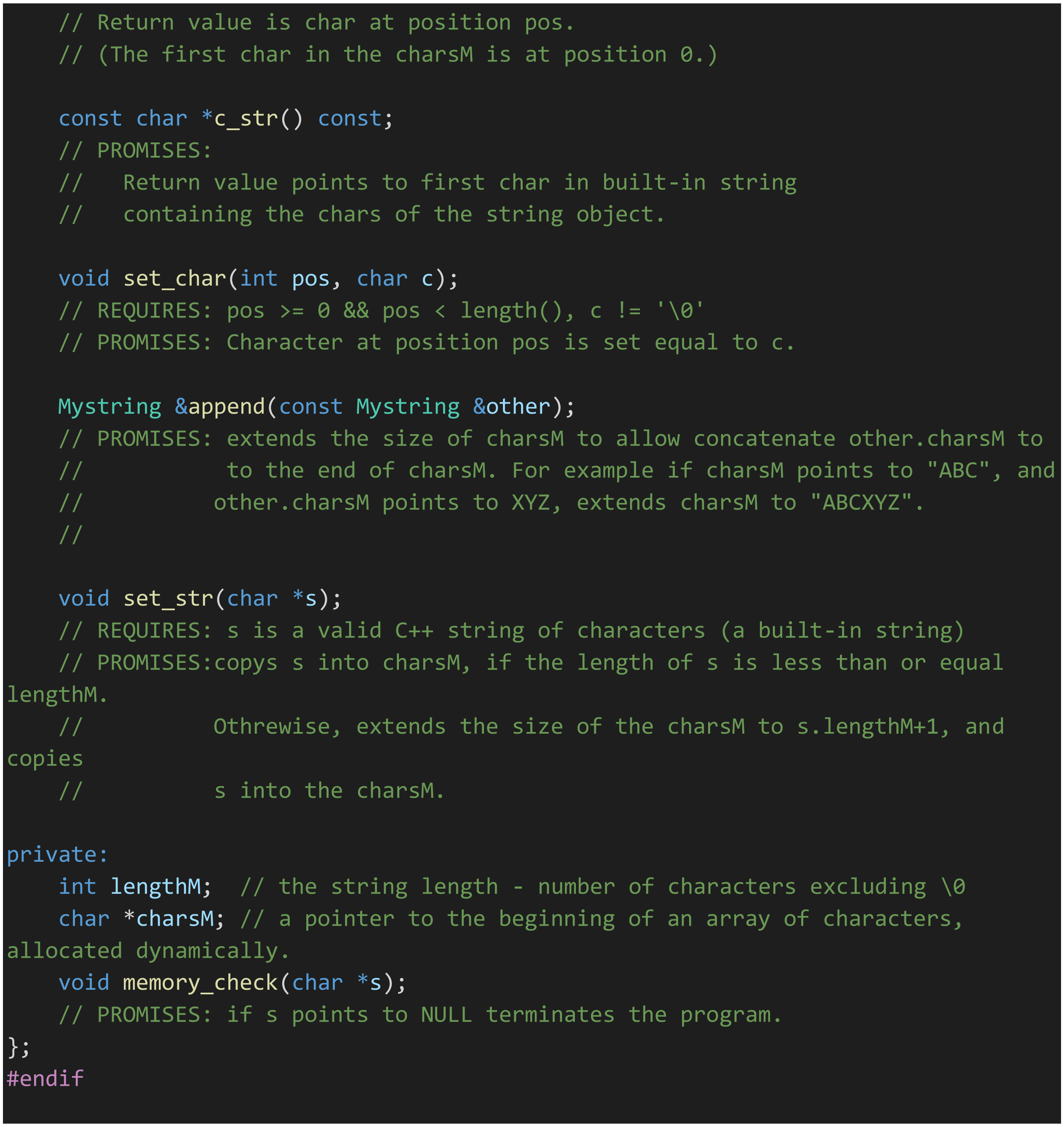
```
    cout << "\n\n***----Finished tests for overloading operators ----------
***\n\n";
}

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';
    }
}
```

Exercise B – Source file mystring_B.h

```
/*
 * File Name:                 mystring_B.h
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise B
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

#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; // is equal to operator
```



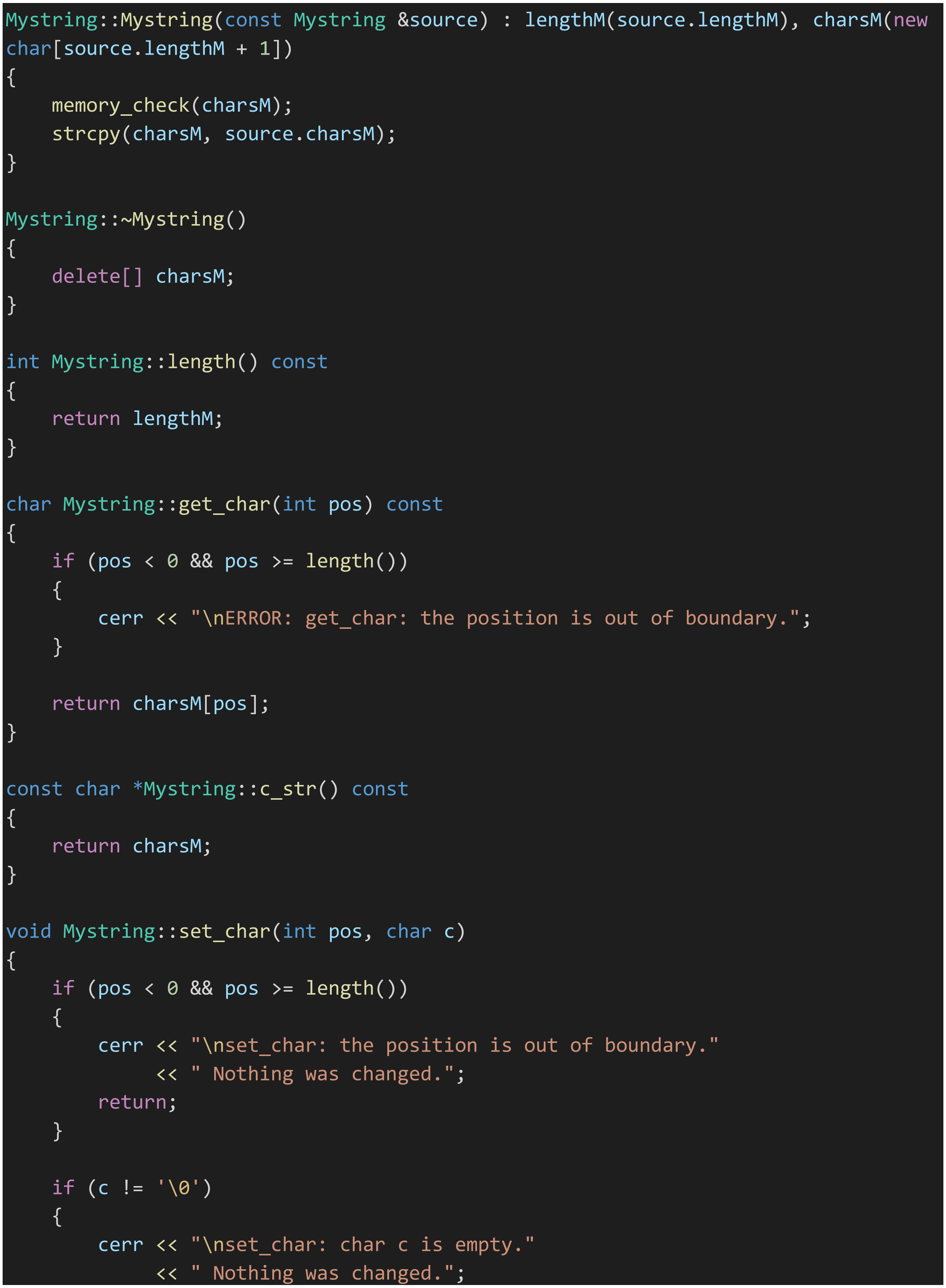
```
    //  REQUIRES: rhs is reference to a Mystring as a source
    //  PROMISES: returns true if this object is lexicographically equal to rhs, otherwise false

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

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

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

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

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

    char &operator[](int i) const;
    //  REQUIRES: 0 <= i <= lengthM
    //  PROMISES: returns the character at index i of string charsM

    friend ostream &operator<<(ostream &os, const Mystring &rhs); // insertion operator
    //  REQUIRES: rhs is reference to a Mystring as a source
    //  PROMISES: prints the charsM to the stdout

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

    char get_char(int pos) const;
    // REQUIRES: pos >= 0 && pos < length()
    // PROMISES:
```



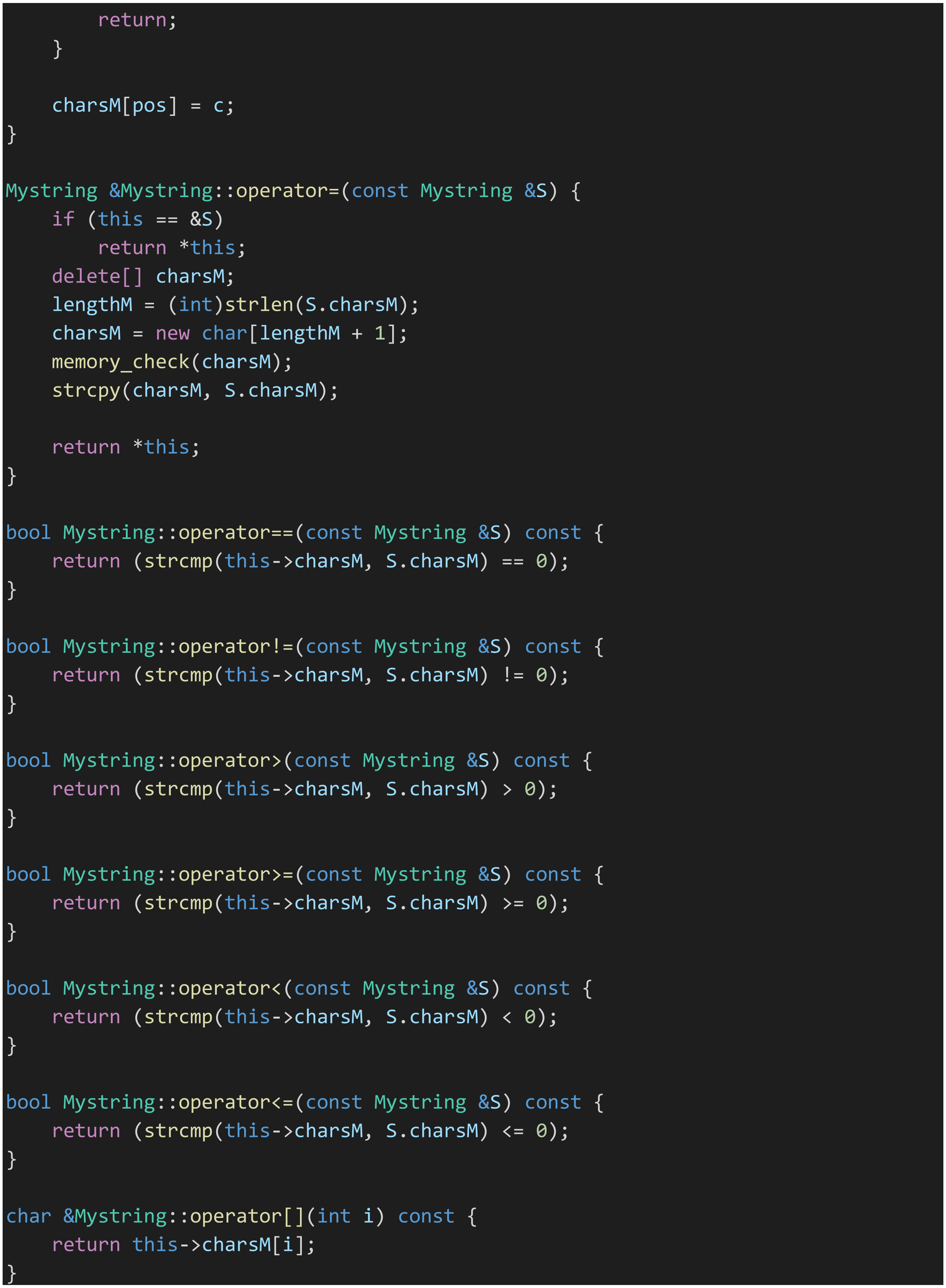
```
    // Return value is char at position pos.
    // (The first char in the charsM is at position 0.)

    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
```

Exercise B – Source file mystring_B.cpp

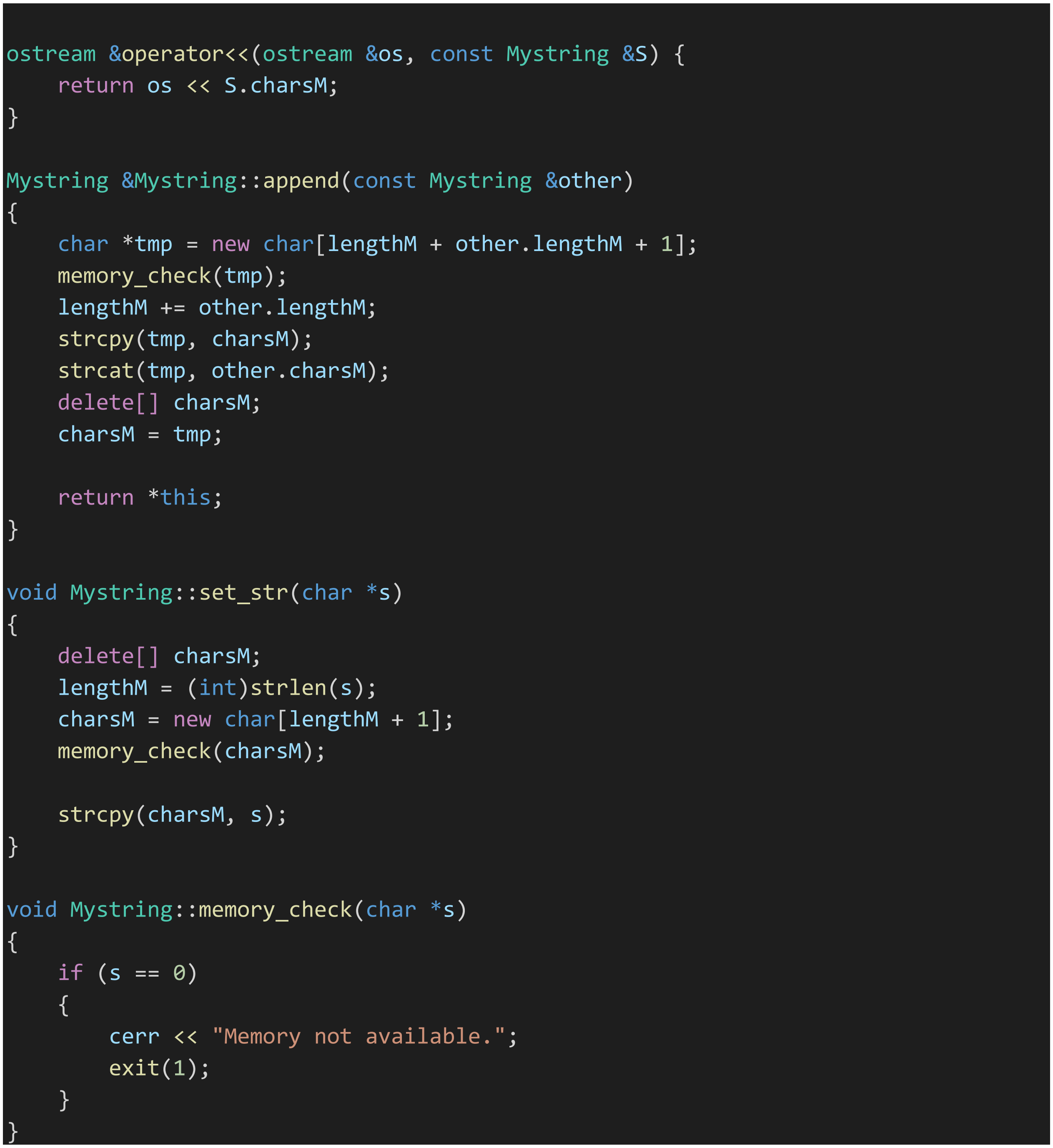
```cpp
/*
 * File Name:                 mystring_B.cpp
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise B
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

#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.";
```



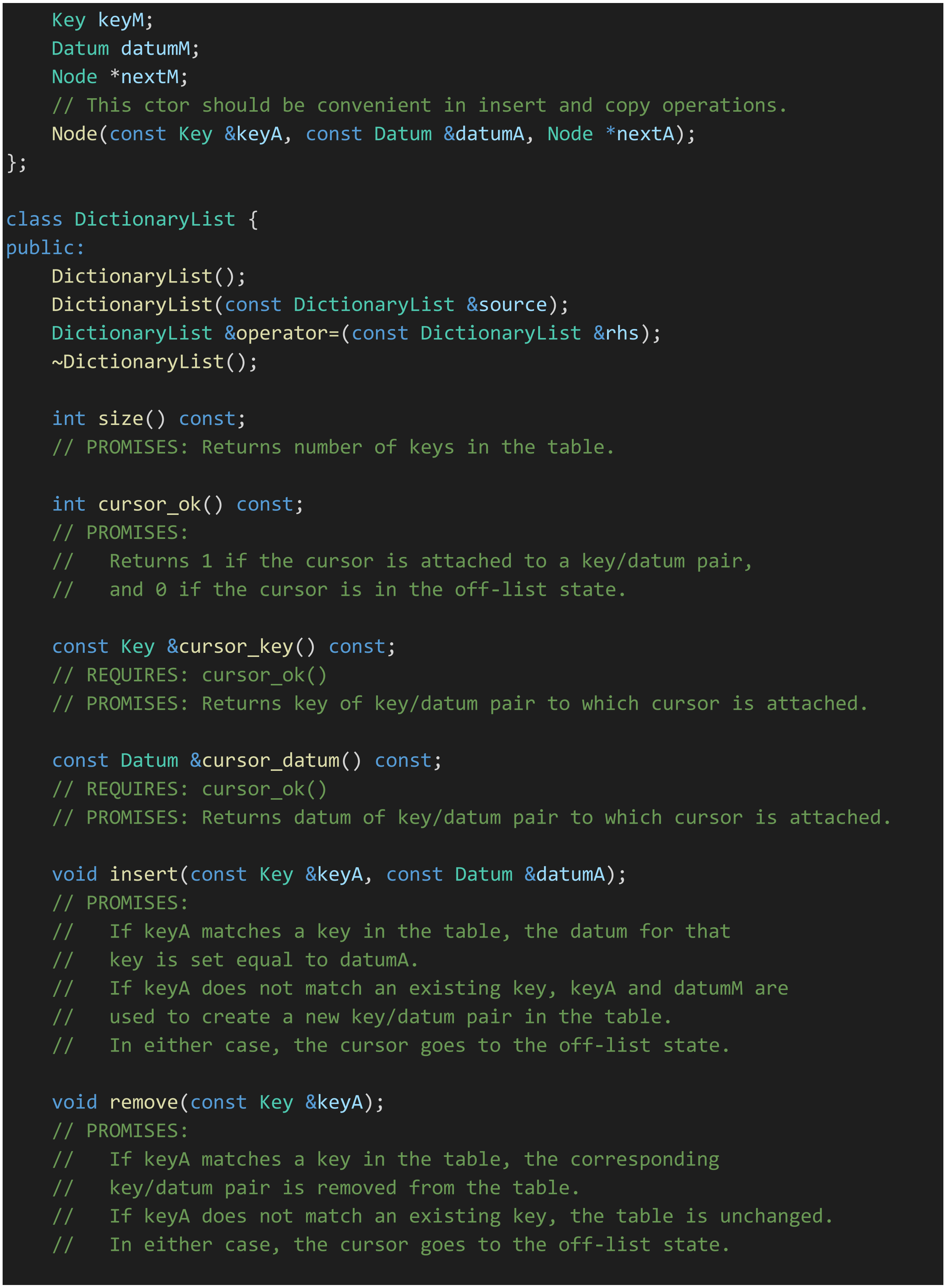
```cpp
        return;
    }

    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 &S) const {
    return (strcmp(this->charsM, S.charsM) == 0);
}

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

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

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

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

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

char &Mystring::operator[](int i) const {
    return this->charsM[i];
}
```



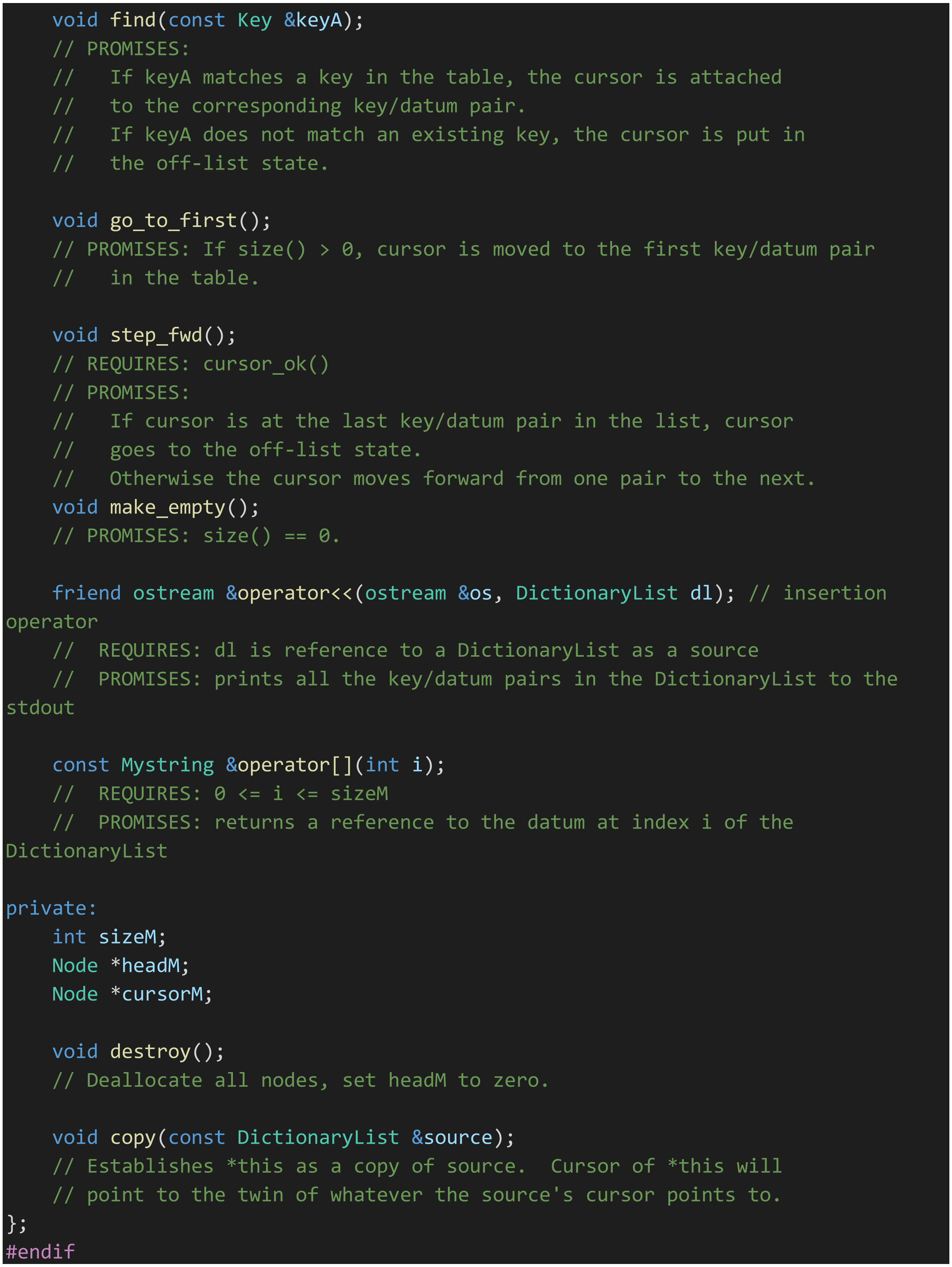
```
ostream &operator<<(ostream &os, const Mystring &S) {
    return os << S.charsM;
}

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);
    }
}
```

Exercise B – Source file dictionaryList.h

```
/*
 * File Name:                 dictionaryList.h
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise A and B
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

#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:
```



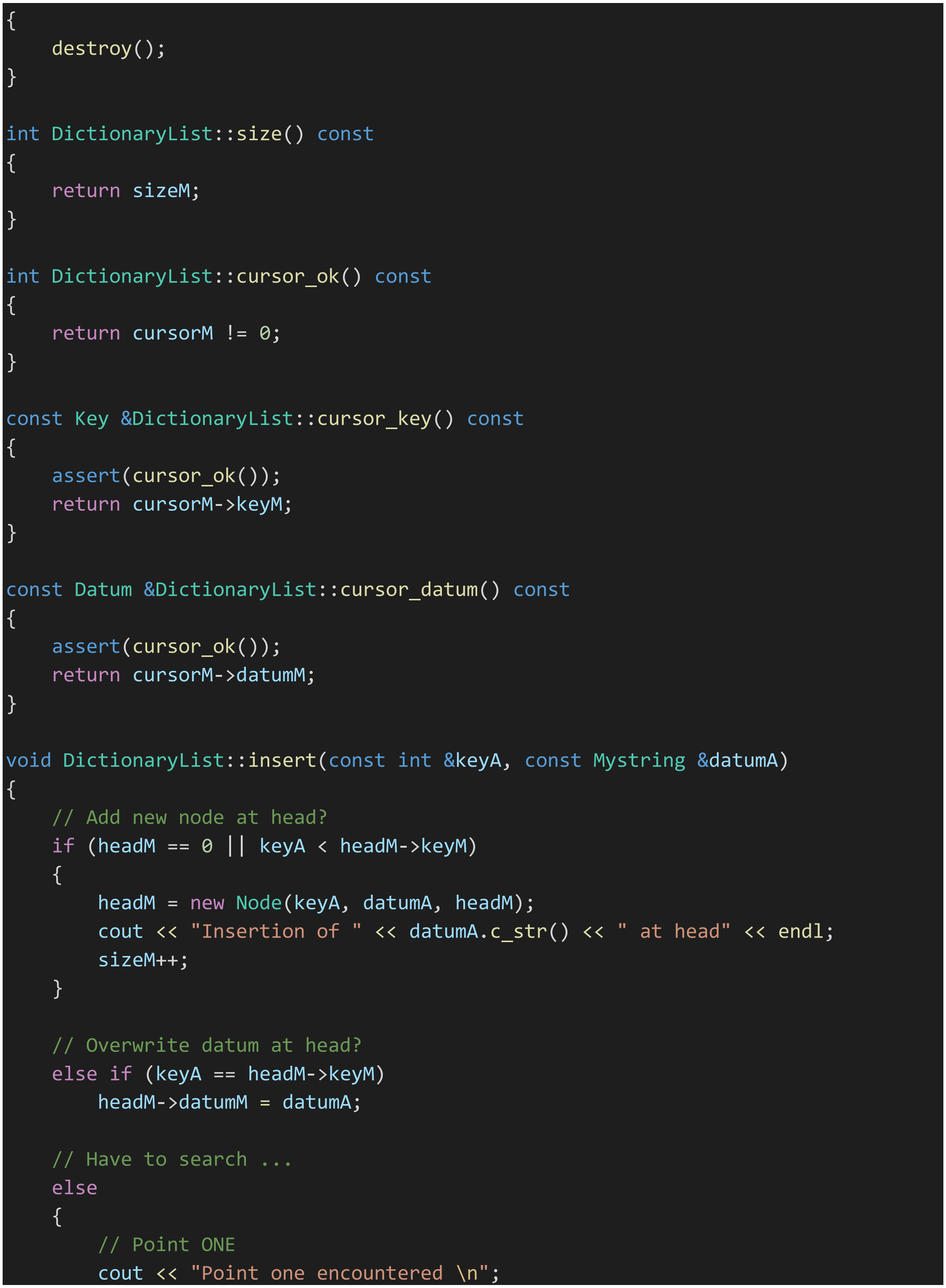
```
    Key keyM;
    Datum datumM;
    Node *nextM;
    // This ctor should be convenient in insert and copy operations.
    Node(const Key &keyA, const Datum &datumA, Node *nextA);
};

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.
```



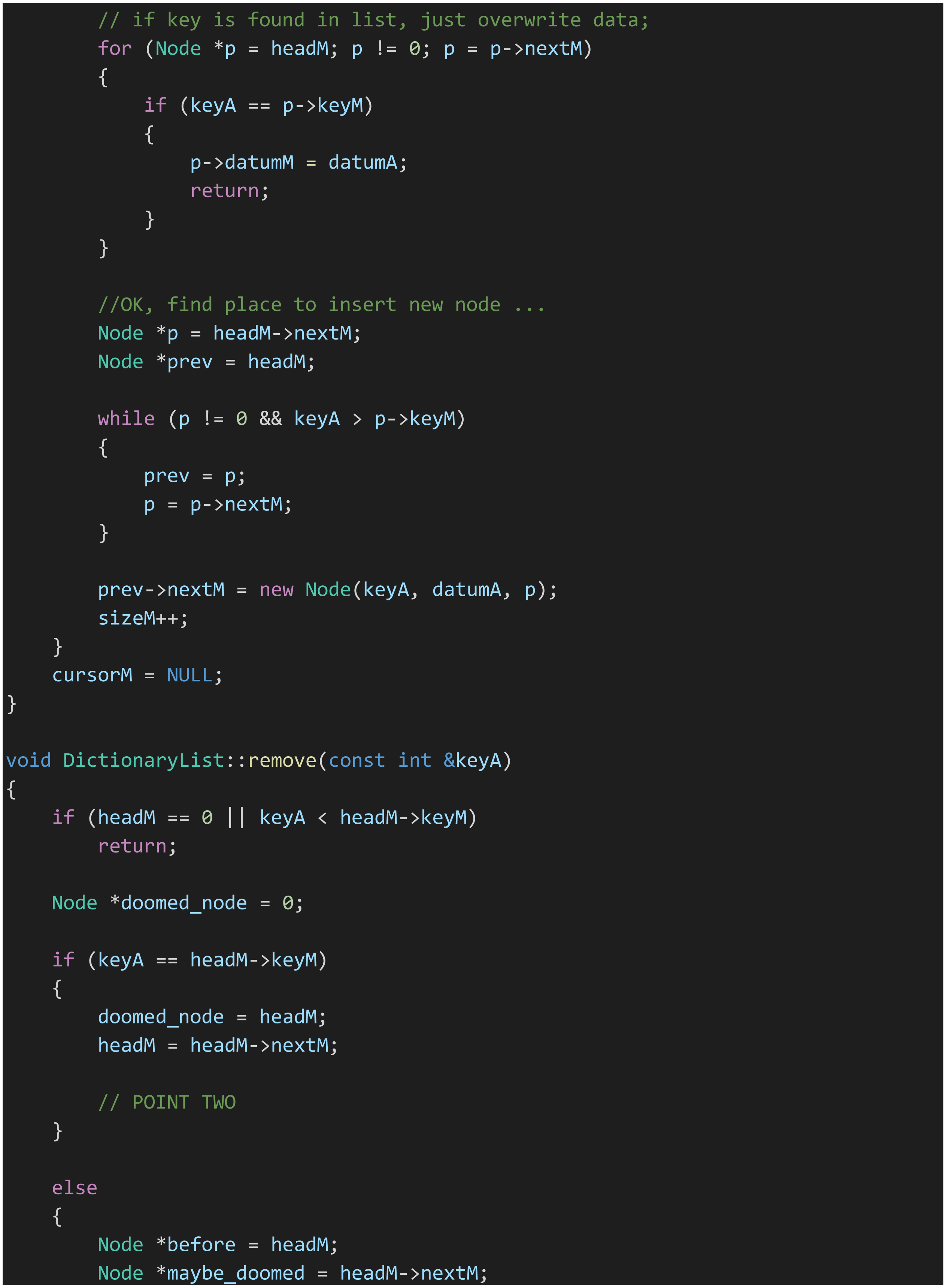
```
    void find(const Key &keyA);
    // PROMISES:
    //   If keyA matches a key in the table, the cursor is attached
    //   to the corresponding key/datum pair.
    //   If keyA does not match an existing key, the cursor is put in
    //   the off-list state.

    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.

    friend ostream &operator<<(ostream &os, DictionaryList dl); // insertion
operator
    //  REQUIRES: dl is reference to a DictionaryList as a source
    //  PROMISES: prints all the key/datum pairs in the DictionaryList to the
stdout

    const Mystring &operator[](int i);
    //  REQUIRES: 0 <= 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
```

Exercise B – Source file dictionaryList.cpp

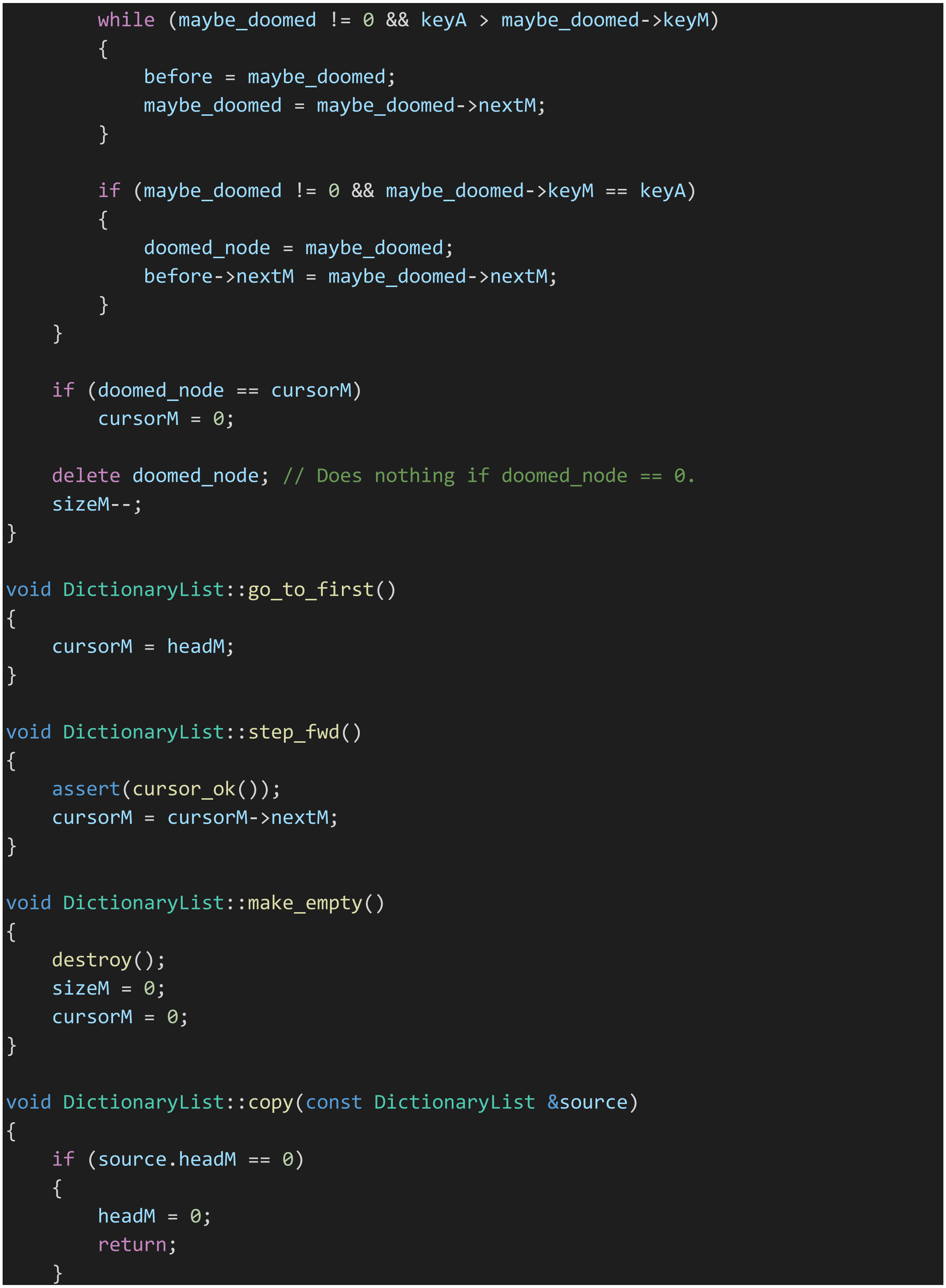
```
/*
 * File Name:                 dictionaryList.cpp
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise A and B
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

#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;
}

DictionaryList::~DictionaryList()
```



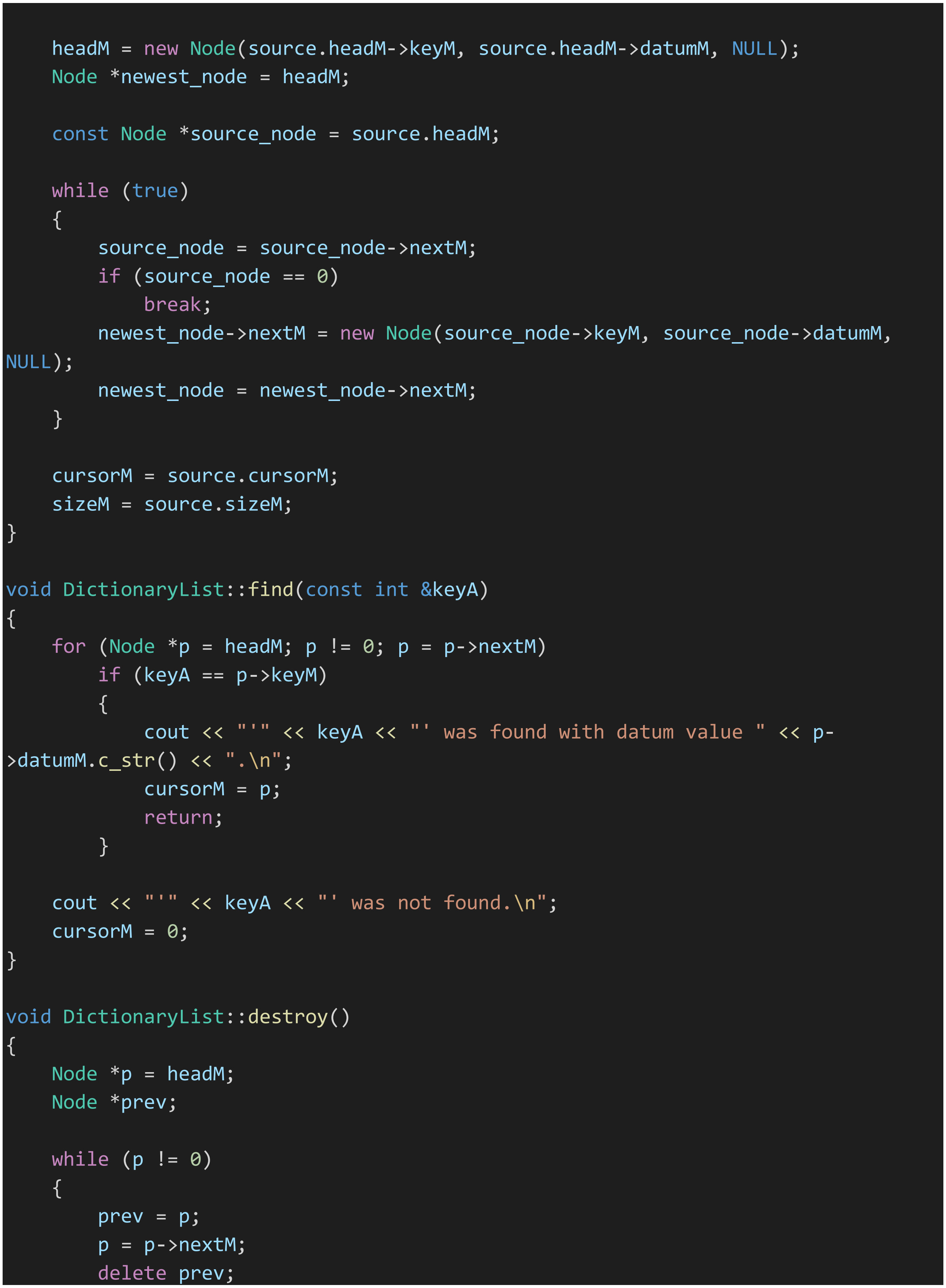
```
{
    destroy();
}

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);
        cout << "Insertion of " << datumA.c_str() << " at head" << endl;
        sizeM++;
    }

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

    // Have to search ...
    else
    {
        // Point ONE
        cout << "Point one encountered \n";
```



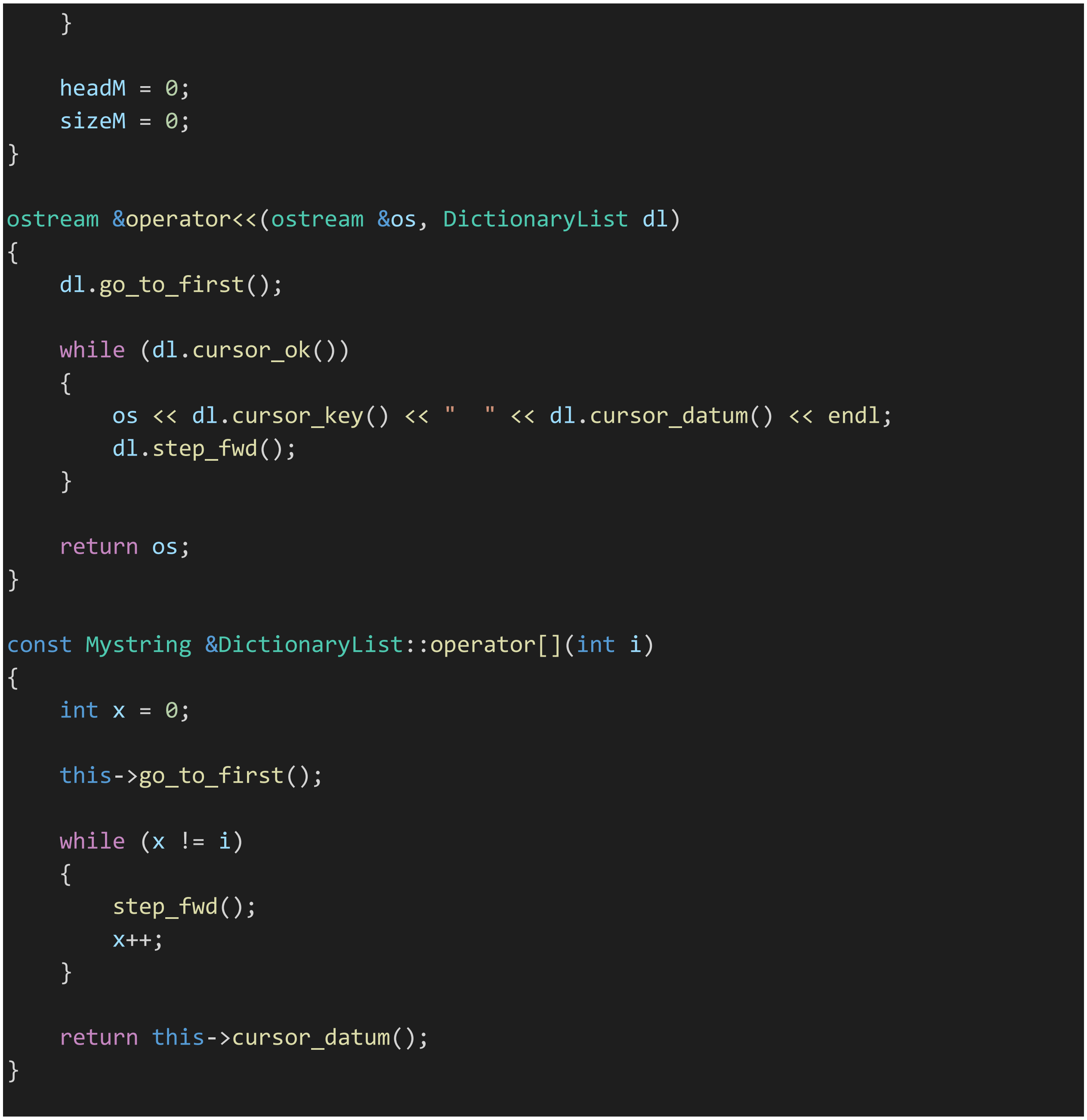
```
        // if key is found in list, just overwrite data;
        for (Node *p = headM; p != 0; p = p->nextM)
        {
            if (keyA == p->keyM)
            {
                p->datumM = datumA;
                return;
            }
        }

        //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;
    }
```

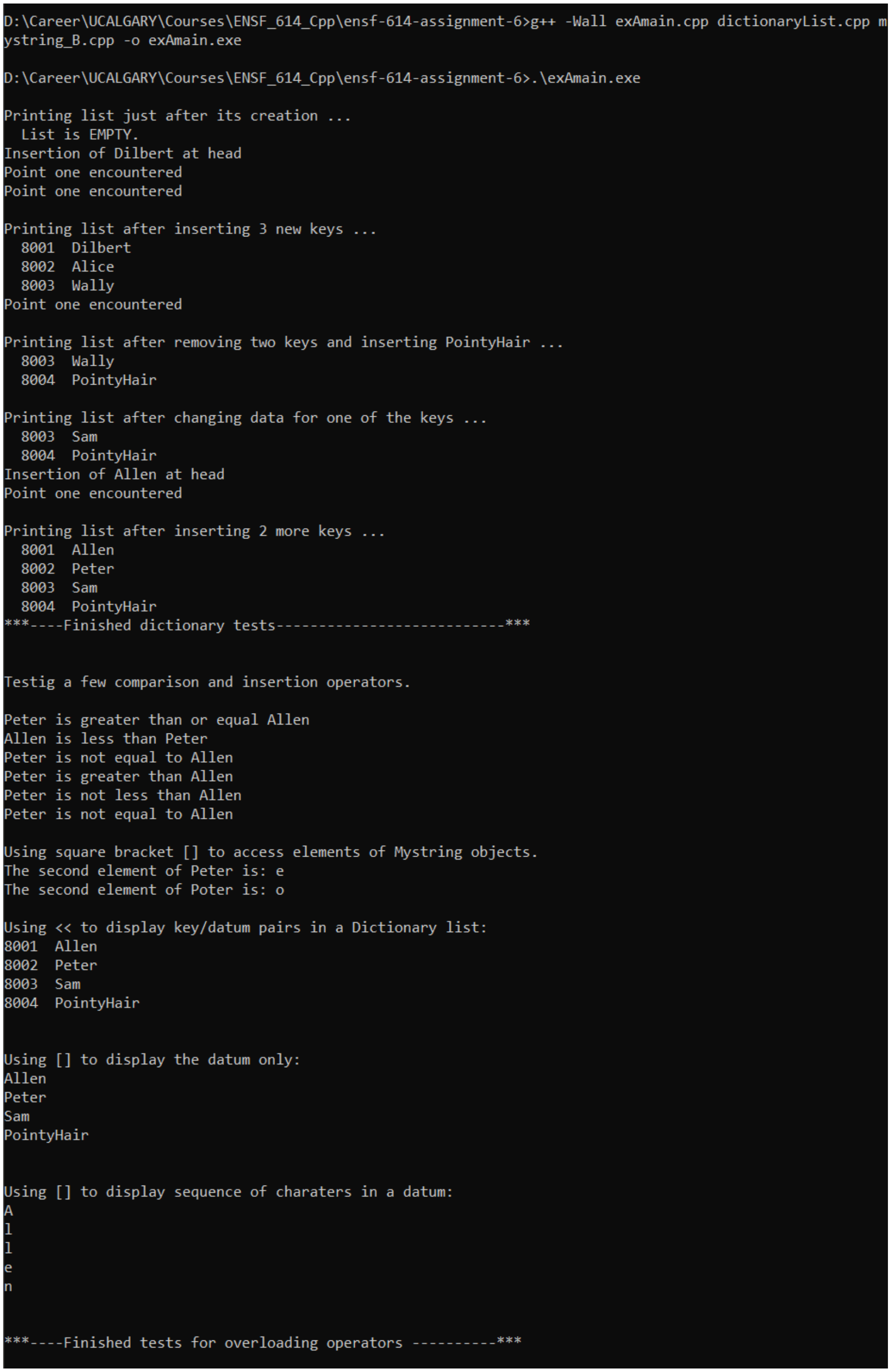


```
    headM = new Node(source.headM->keyM, source.headM->datumM, NULL);
    Node *newest_node = headM;

    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;
}

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)
{
    int x = 0;

    this->go_to_first();

    while (x != i)
    {
        step_fwd();
        x++;
    }

    return this->cursor_datum();
}
```

Exercise B – Program Output



```
D:\Career\UCALGARY\Courses\ENSF_614_Cpp\ensf-614-assignment-6>g++ -Wall exAmain.cpp dictionaryList.cpp m
ystring_B.cpp -o exAmain.exe

D:\Career\UCALGARY\Courses\ENSF_614_Cpp\ensf-614-assignment-6>.\exAmain.exe

Printing list just after its creation ...
  List is EMPTY.
Insertion of Dilbert at head
Point one encountered
Point one encountered

Printing list after inserting 3 new keys ...
  8001  Dilbert
  8002  Alice
  8003  Wally
Point one encountered

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
Insertion of Allen at head
Point one encountered

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 file Item.java

```
/*
 * File Name:                 Item.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise C and D
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

class Item<E extends Number & Comparable<E>> {
    private E item;

    public Item(E value) {
        item = value;
    }

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

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

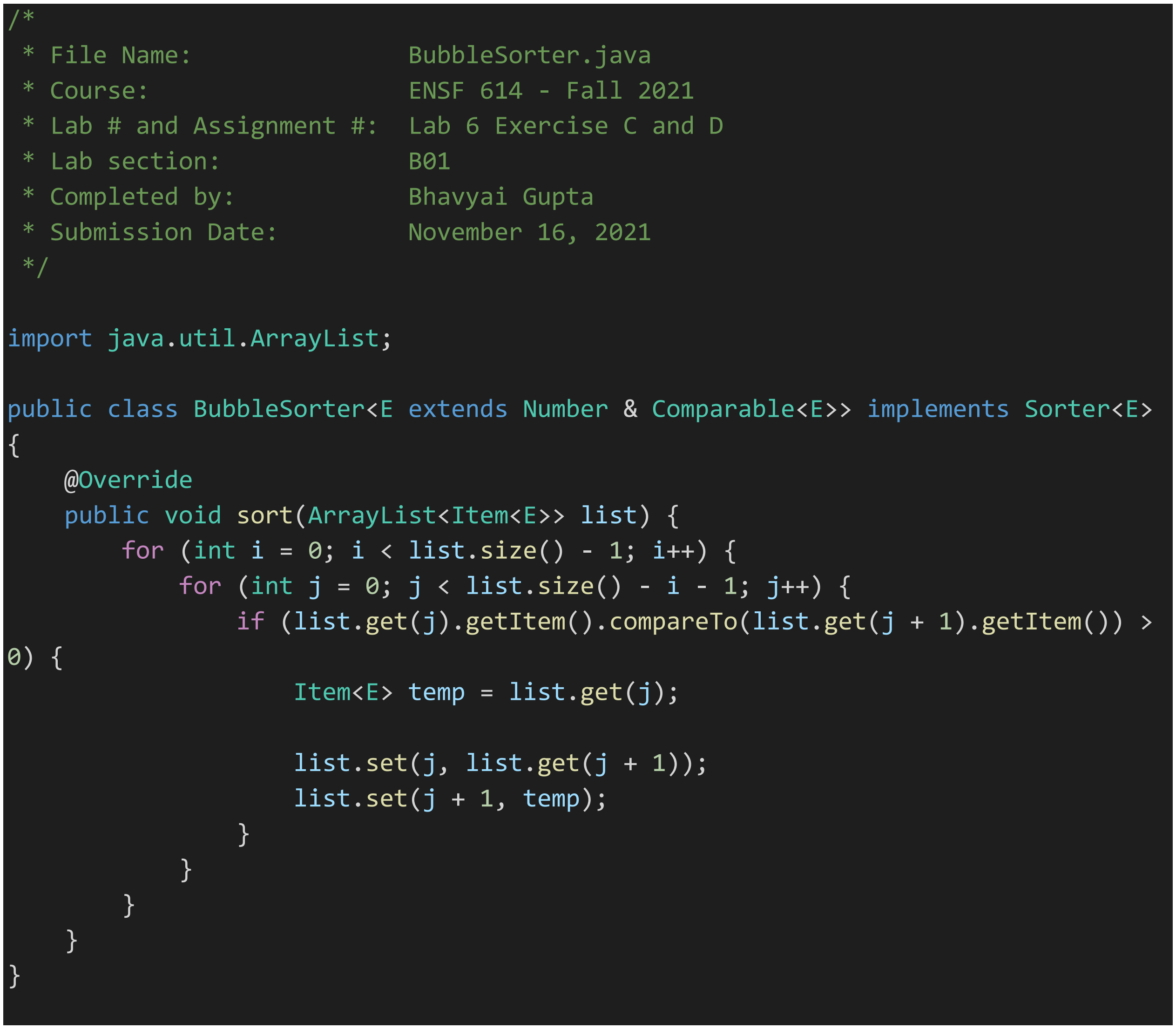
Exercise C and D – Source file MyVector.java

```
/*
 * File Name:                 MyVector.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise C and D
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

import java.util.ArrayList;

public class MyVector<E extends Number & Comparable<E>> {
    private ArrayList<Item<E>> storageM;
    private Sorter<E> sorter;

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

    MyVector(ArrayList<Item<E>> arr) {
        this.storageM = new ArrayList<>(arr.size());

        for (Item<E> i : arr) {
            this.storageM.add(i);
        }
    }

    public void add(Item<E> value) {
        this.storageM.add(value);
    }

    public void setSortStrategy(Sorter<E> s) {
        this.sorter = s;
    }

    public void performSort() {
        this.sorter.sort((this.storageM));
    }

    public void display() {
        for(Item<E> i : this.storageM) {
            if(i.getItem() instanceof Integer) {
                System.out.printf("%d ", i.getItem());
            }

            else {
                System.out.printf("%.2f ", i.getItem());
            }
        }
    }
}
```

Exercise C and D – Source file Sorter.java

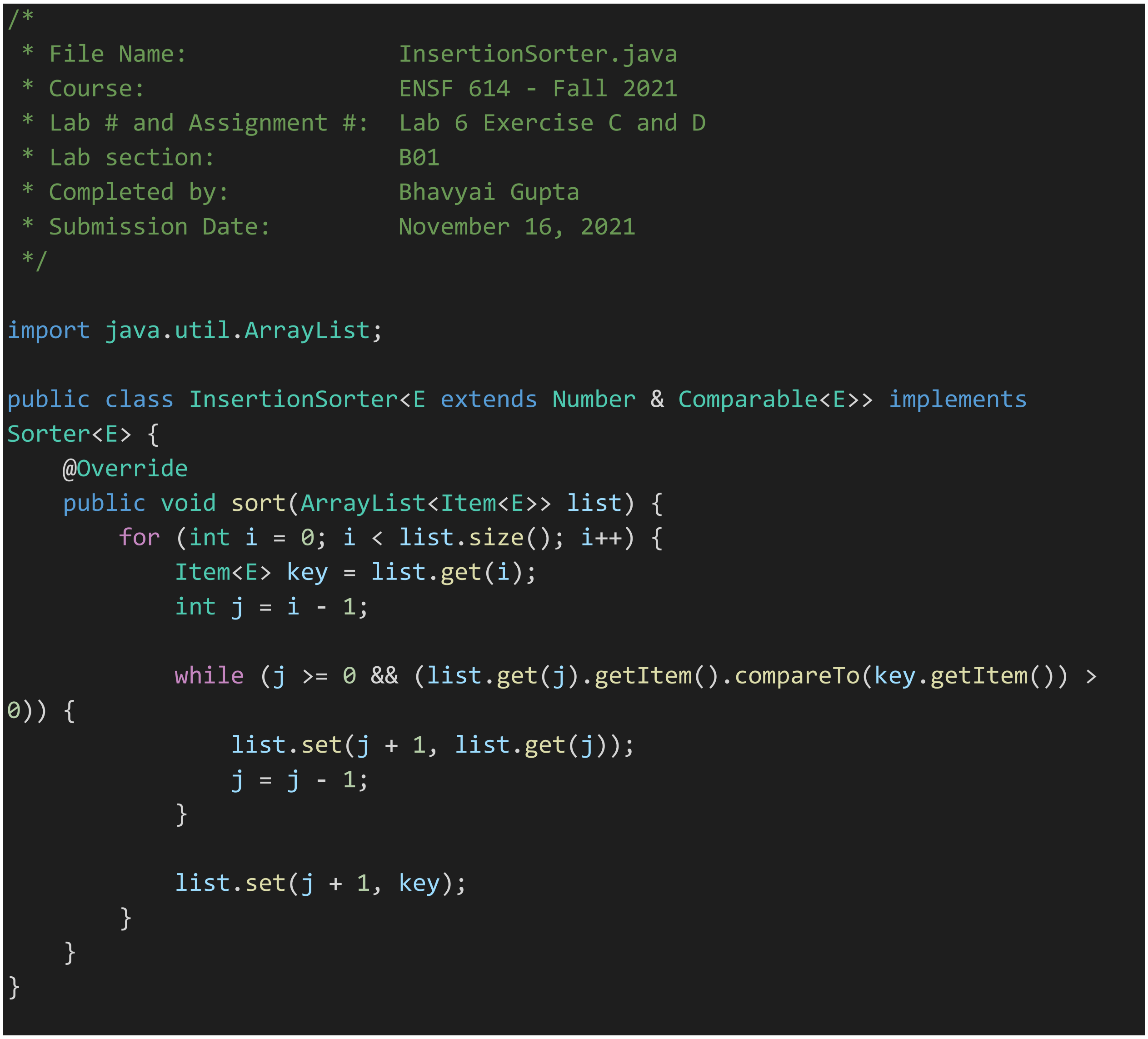
```java
/*
 * File Name:                Sorter.java
 * Course:                   ENSF 614 - Fall 2021
 * Lab # and Assignment #:   Lab 6 Exercise C and D
 * Lab section:              B01
 * Completed by:             Bhavyai Gupta
 * Submission Date:          November 16, 2021
 */

import java.util.ArrayList;

public interface Sorter<E extends Number & Comparable<E>> {
    public void sort(ArrayList<Item<E>> list);
}
```

Exercise C and D – Source file BubbleSorter.java



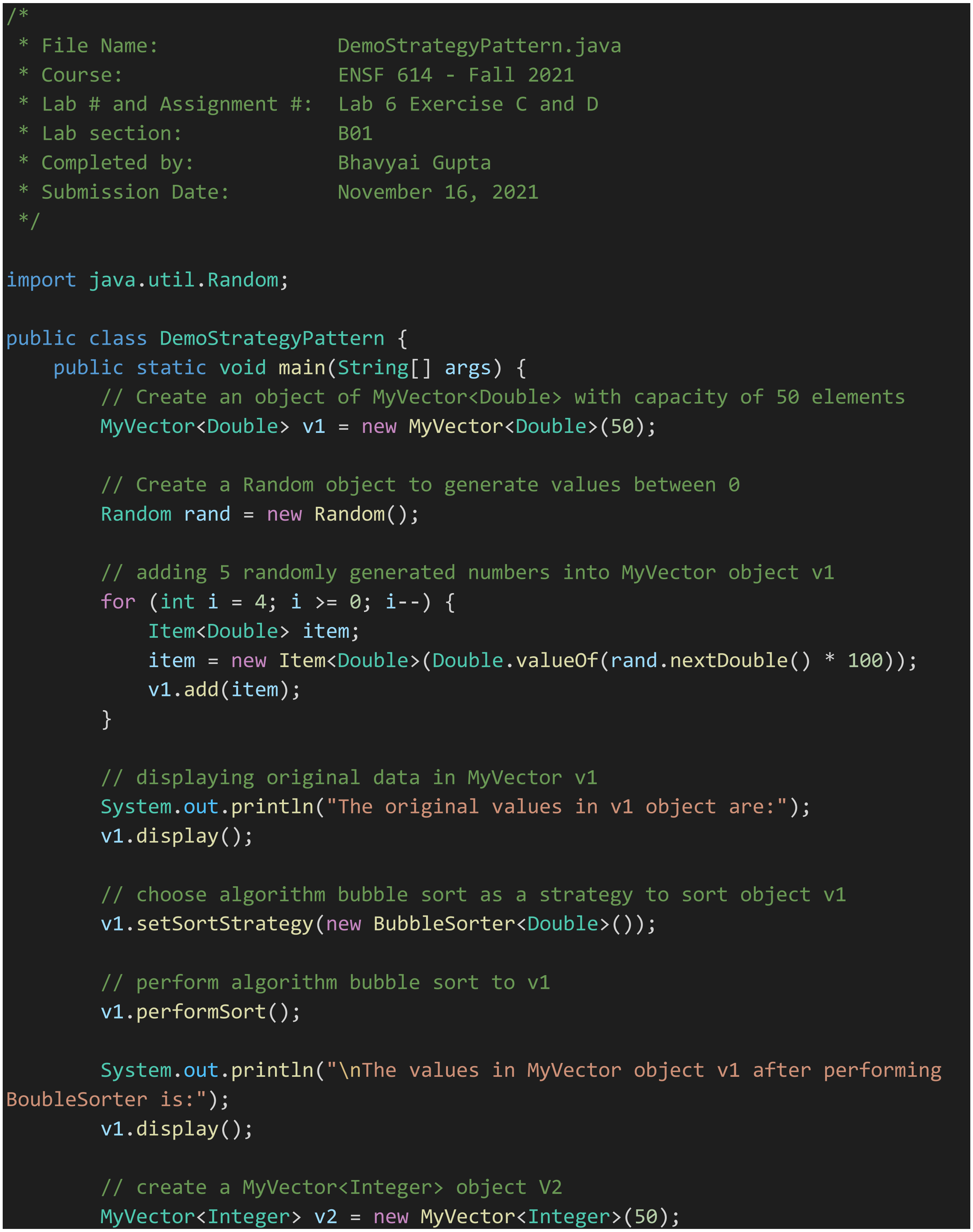
```java
/*
 * File Name:                 BubbleSorter.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise C and D
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

import java.util.ArrayList;

public class BubbleSorter<E extends Number & Comparable<E>> implements Sorter<E>
{
    @Override
    public void sort(ArrayList<Item<E>> list) {
        for (int i = 0; i < list.size() - 1; i++) {
            for (int j = 0; j < list.size() - i - 1; j++) {
                if (list.get(j).getItem().compareTo(list.get(j + 1).getItem()) >
0) {
                    Item<E> temp = list.get(j);

                    list.set(j, list.get(j + 1));
                    list.set(j + 1, temp);
                }
            }
        }
    }
}
```

Exercise C and D – Source file InsertionSorter.java



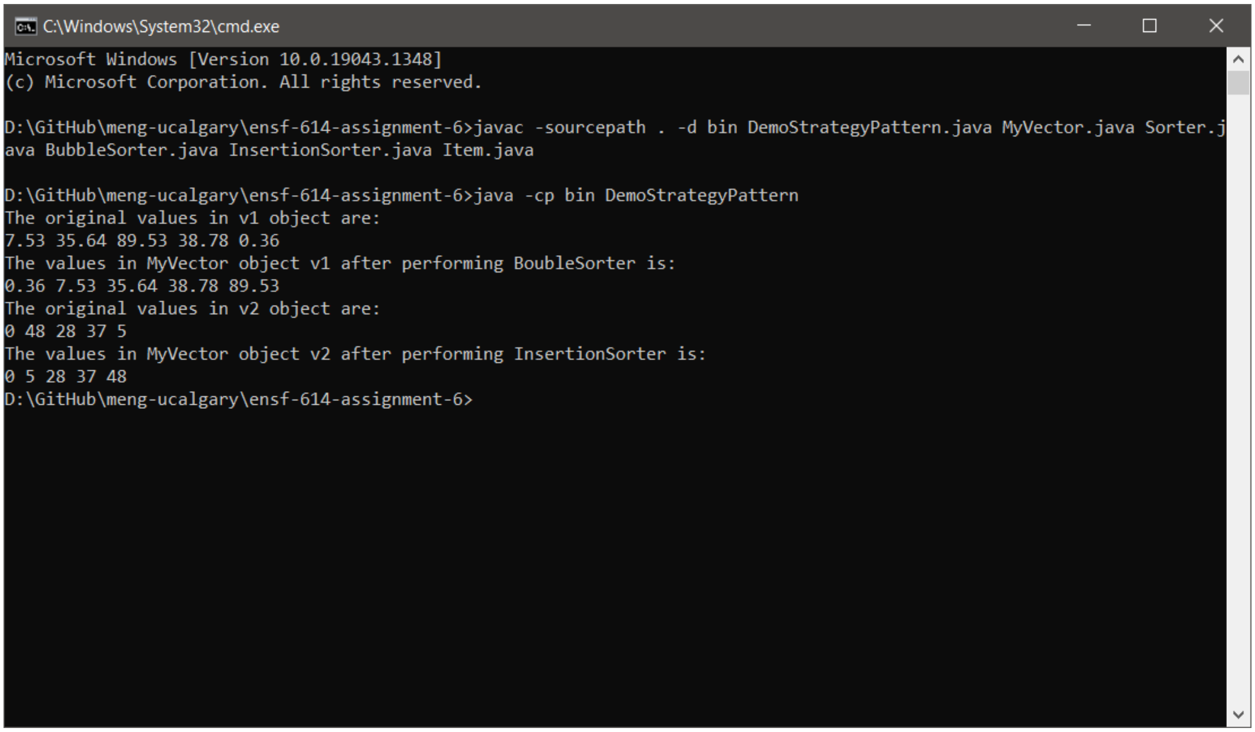
```
/*
 * File Name:               InsertionSorter.java
 * Course:                  ENSF 614 - Fall 2021
 * Lab # and Assignment #:  Lab 6 Exercise C and D
 * Lab section:             B01
 * Completed by:            Bhavyai Gupta
 * Submission Date:         November 16, 2021
 */

import java.util.ArrayList;

public class InsertionSorter<E extends Number & Comparable<E>> implements
Sorter<E> {
    @Override
    public void sort(ArrayList<Item<E>> list) {
        for (int i = 0; i < list.size(); i++) {
            Item<E> key = list.get(i);
            int j = i - 1;

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

            list.set(j + 1, key);
        }
    }
}
```

Exercise C – Source file DemoStrategyPattern.java



```
/*
 * File Name:                DemoStrategyPattern.java
 * Course:                   ENSF 614 - Fall 2021
 * Lab # and Assignment #:   Lab 6 Exercise C and D
 * Lab section:              B01
 * Completed by:             Bhavyai Gupta
 * Submission Date:          November 16, 2021
 */

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<Double> v1 = new MyVector<Double>(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<Double> item;
            item = new Item<Double>(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<Double>());

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

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

Exercise C – Program Output



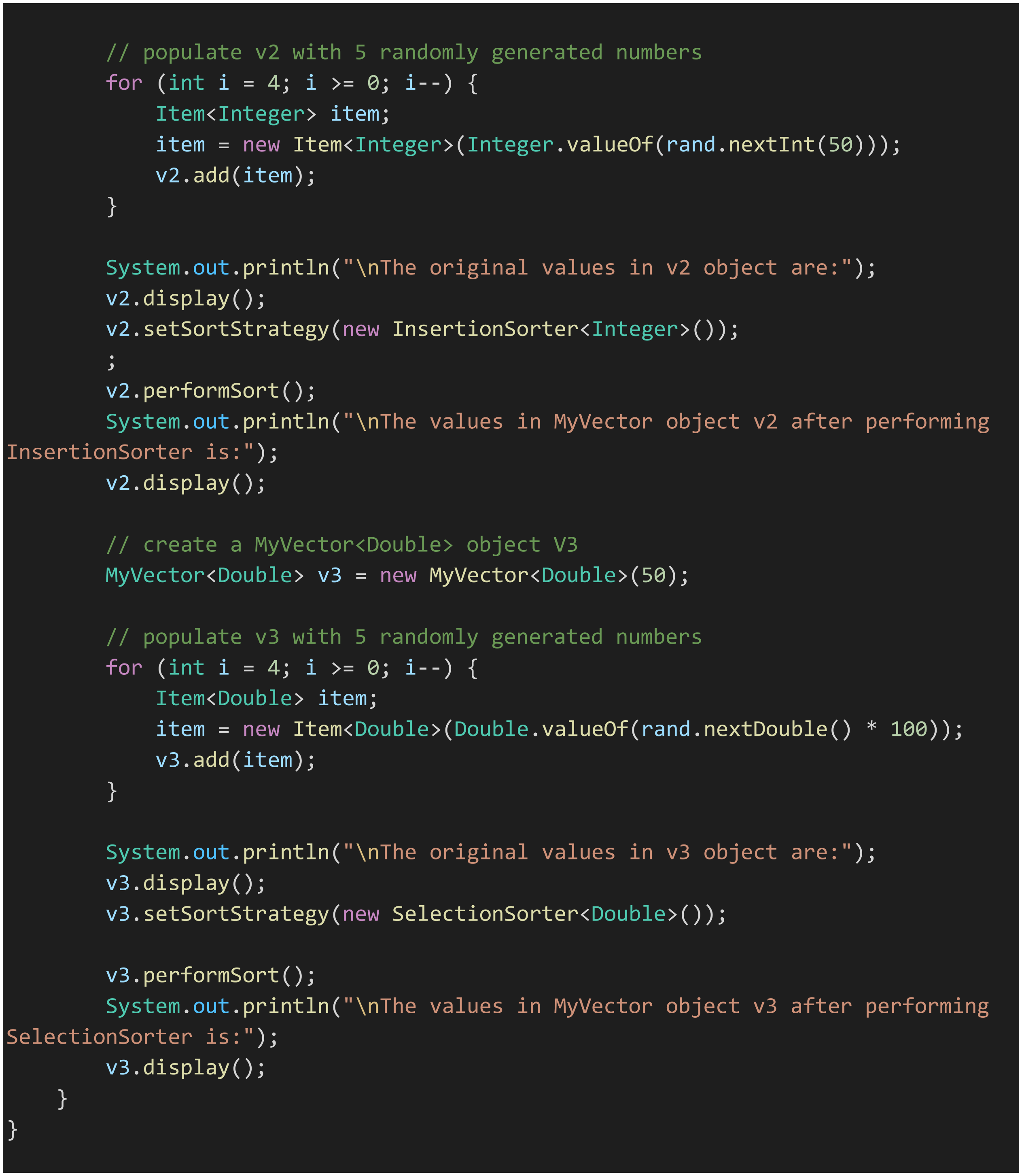
```
Microsoft Windows [Version 10.0.19043.1348]
(c) Microsoft Corporation. All rights reserved.

D:\GitHub\meng-ucalgary\ensf-614-assignment-6>javac -sourcepath . -d bin DemoStrategyPattern.java MyVector.java Sorter.j
ava BubbleSorter.java InsertionSorter.java Item.java

D:\GitHub\meng-ucalgary\ensf-614-assignment-6>java -cp bin DemoStrategyPattern
The original values in v1 object are:
7.53 35.64 89.53 38.78 0.36
The values in MyVector object v1 after performing BoubleSorter is:
0.36 7.53 35.64 38.78 89.53
The original values in v2 object are:
0 48 28 37 5
The values in MyVector object v2 after performing InsertionSorter is:
0 5 28 37 48
D:\GitHub\meng-ucalgary\ensf-614-assignment-6>
```

Exercise D – Source file SelectionSorter.java

```java
/*
 * File Name:                 SelectionSorter.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise C and D
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

import java.util.ArrayList;

public class SelectionSorter<E extends Number & Comparable<E>> implements
Sorter<E> {
    @Override
    public void sort(ArrayList<Item<E>> list) {
        for (int i = 0; i < list.size() - 1; i++) {
            int min_idx = i;

            for (int j = i + 1; j < list.size(); j++) {
                if (list.get(j).getItem().compareTo(list.get(min_idx).getItem())
< 0) {
                    min_idx = j;
                }
            }

            Item<E> temp = list.get(min_idx);
            list.set(min_idx, list.get(i));
            list.set(i, temp);
        }
    }
}
```

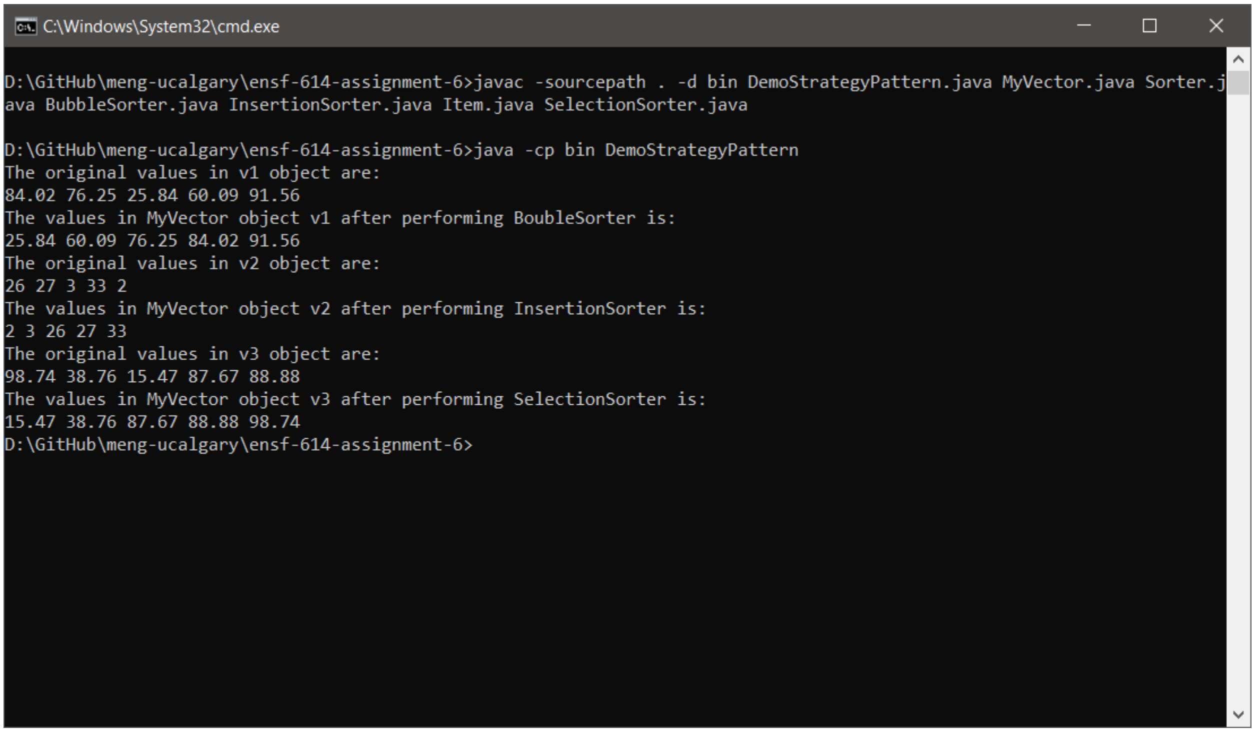
Exercise D – Source file DemoStrategyPattern.java

```
/*
 * File Name:               DemoStrategyPattern.java
 * Course:                  ENSF 614 - Fall 2021
 * Lab # and Assignment #:  Lab 6 Exercise C and D
 * Lab section:             B01
 * Completed by:            Bhavyai Gupta
 * Submission Date:         November 16, 2021
 */

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<Double> v1 = new MyVector<Double>(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<Double> item;
            item = new Item<Double>(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<Double>());

        // 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<Integer> v2 = new MyVector<Integer>(50);
```



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

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

        // create a MyVector<Double> object V3
        MyVector<Double> v3 = new MyVector<Double>(50);

        // populate v3 with 5 randomly generated numbers
        for (int i = 4; i >= 0; i--) {
            Item<Double> item;
            item = new Item<Double>(Double.valueOf(rand.nextDouble() * 100));
            v3.add(item);
        }

        System.out.println("\nThe original values in v3 object are:");
        v3.display();
        v3.setSortStrategy(new SelectionSorter<Double>());

        v3.performSort();
        System.out.println("\nThe values in MyVector object v3 after performing 
SelectionSorter is:");
        v3.display();
    }
}
```

Exercise D – Program Output



```
D:\GitHub\meng-ucalgary\ensf-614-assignment-6>javac -sourcepath . -d bin DemoStrategyPattern.java MyVector.java Sorter.j
ava BubbleSorter.java InsertionSorter.java Item.java SelectionSorter.java

D:\GitHub\meng-ucalgary\ensf-614-assignment-6>java -cp bin DemoStrategyPattern
The original values in v1 object are:
84.02 76.25 25.84 60.09 91.56
The values in MyVector object v1 after performing BoubleSorter is:
25.84 60.09 76.25 84.02 91.56
The original values in v2 object are:
26 27 3 33 2
The values in MyVector object v2 after performing InsertionSorter is:
2 3 26 27 33
The original values in v3 object are:
98.74 38.76 15.47 87.67 88.88
The values in MyVector object v3 after performing SelectionSorter is:
15.47 38.76 87.67 88.88 98.74
D:\GitHub\meng-ucalgary\ensf-614-assignment-6>
```

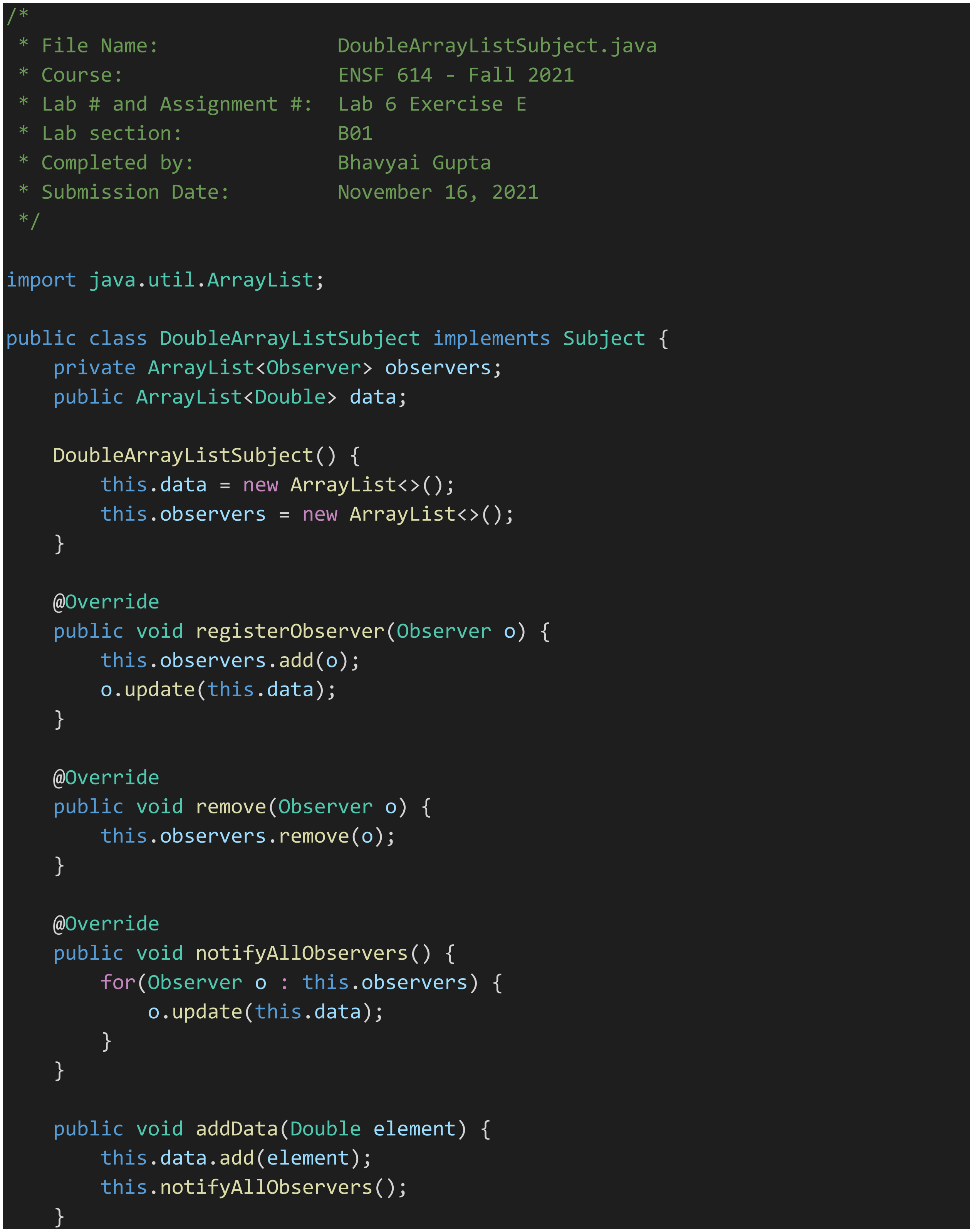
Exercise E – Source file ObserverPatternController.java

```java
/*
 * File Name:                 ObserverPatternController.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise E
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

public class ObserverPatternController {
    public static void main(String[] s) {
        double[] arr = { 10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11, 23, 34, 55
};
        System.out.println("Creating object mydata with an empty list -- no
data:");
        DoubleArrayListSubject mydata = new DoubleArrayListSubject();
        System.out.println("Expected to print: Empty List ...");
        mydata.display();
        mydata.populate(arr);
        System.out.println("mydata object is populated with: 10, 20, 33, 44, 50,
30, 60, 70, 80, 10, 11, 23, 34, 55 ");
        System.out.print("Now, creating three observer objects: ht, vt, and hl
");
        System.out.println("\nwhich are immediately notified of existing data
with different views.");
        ThreeColumnTable_Observer ht = new ThreeColumnTable_Observer(mydata);
        FiveRowsTable_Observer vt = new FiveRowsTable_Observer(mydata);
        OneRow_Observer hl = new OneRow_Observer(mydata);
        System.out.println("\n\nChanging the third value from 33, to 66 -- (All
views must show this change):");
        mydata.setData(66.0, 2);
        System.out.println("\n\nAdding a new value to the end of the list -- (All
views must show this change)");
        mydata.addData(1000.0);
        System.out.println("\n\nNow removing two observers from the list:");
        mydata.remove(ht);
        mydata.remove(vt);
        System.out.println("Only the remained observer (One Row ), is
notified.");
        mydata.addData(2000.0);
        System.out.println("\n\nNow removing the last observer from the list:");
        mydata.remove(hl);
        System.out.println("\nAdding a new value the end of the list:");
        mydata.addData(3000.0);
        System.out.println("Since there is no observer -- nothing is displayed
...");
        System.out.print("\nNow, creating a new Three-Column observer that will
be notified of existing data:");
        ht = new ThreeColumnTable_Observer(mydata);
    }
}
```

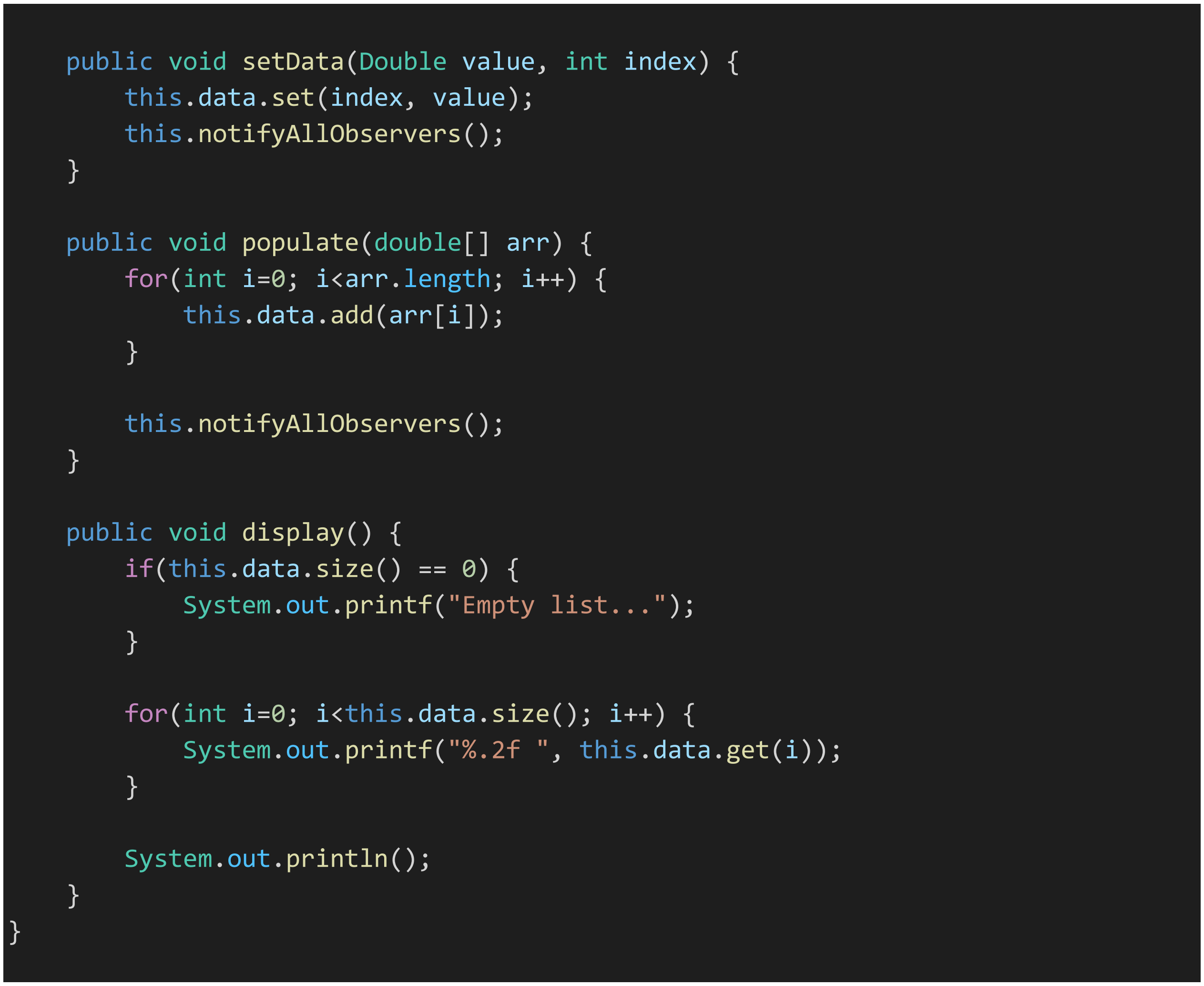
Exercise E – Source file Subject.java

```java
/*
 * File Name:                 Subject.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise E
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

public interface Subject {
    public void registerObserver(Observer o);

    public void remove(Observer o);

    public void notifyAllObservers();
}
```

Exercise E – Source file DoubleArrayListSubject.java



```java
/*
 * File Name:               DoubleArrayListSubject.java
 * Course:                  ENSF 614 - Fall 2021
 * Lab # and Assignment #:  Lab 6 Exercise E
 * Lab section:             B01
 * Completed by:            Bhavyai Gupta
 * Submission Date:         November 16, 2021
 */

import java.util.ArrayList;

public class DoubleArrayListSubject implements Subject {
    private ArrayList<Observer> observers;
    public ArrayList<Double> data;

    DoubleArrayListSubject() {
        this.data = new ArrayList<>();
        this.observers = new ArrayList<>();
    }

    @Override
    public void registerObserver(Observer o) {
        this.observers.add(o);
        o.update(this.data);
    }

    @Override
    public void remove(Observer o) {
        this.observers.remove(o);
    }

    @Override
    public void notifyAllObservers() {
        for(Observer o : this.observers) {
            o.update(this.data);
        }
    }

    public void addData(Double element) {
        this.data.add(element);
        this.notifyAllObservers();
    }
```



```java
    public void setData(Double value, int index) {
        this.data.set(index, value);
        this.notifyAllObservers();
    }

    public void populate(double[] arr) {
        for(int i=0; i<arr.length; i++) {
            this.data.add(arr[i]);
        }

        this.notifyAllObservers();
    }

    public void display() {
        if(this.data.size() == 0) {
            System.out.printf("Empty list...");
        }

        for(int i=0; i<this.data.size(); i++) {
            System.out.printf("%.2f ", this.data.get(i));
        }

        System.out.println();
    }
}
```

Exercise E – Source file Observer.java

```java
/*
 * File Name:                Observer.java
 * Course:                   ENSF 614 - Fall 2021
 * Lab # and Assignment #:   Lab 6 Exercise E
 * Lab section:              B01
 * Completed by:             Bhavyai Gupta
 * Submission Date:          November 16, 2021
 */

import java.util.ArrayList;

public interface Observer {
    public void update(ArrayList<Double> arr);
}
```

Exercise E – Source file OneRow_Observer.java

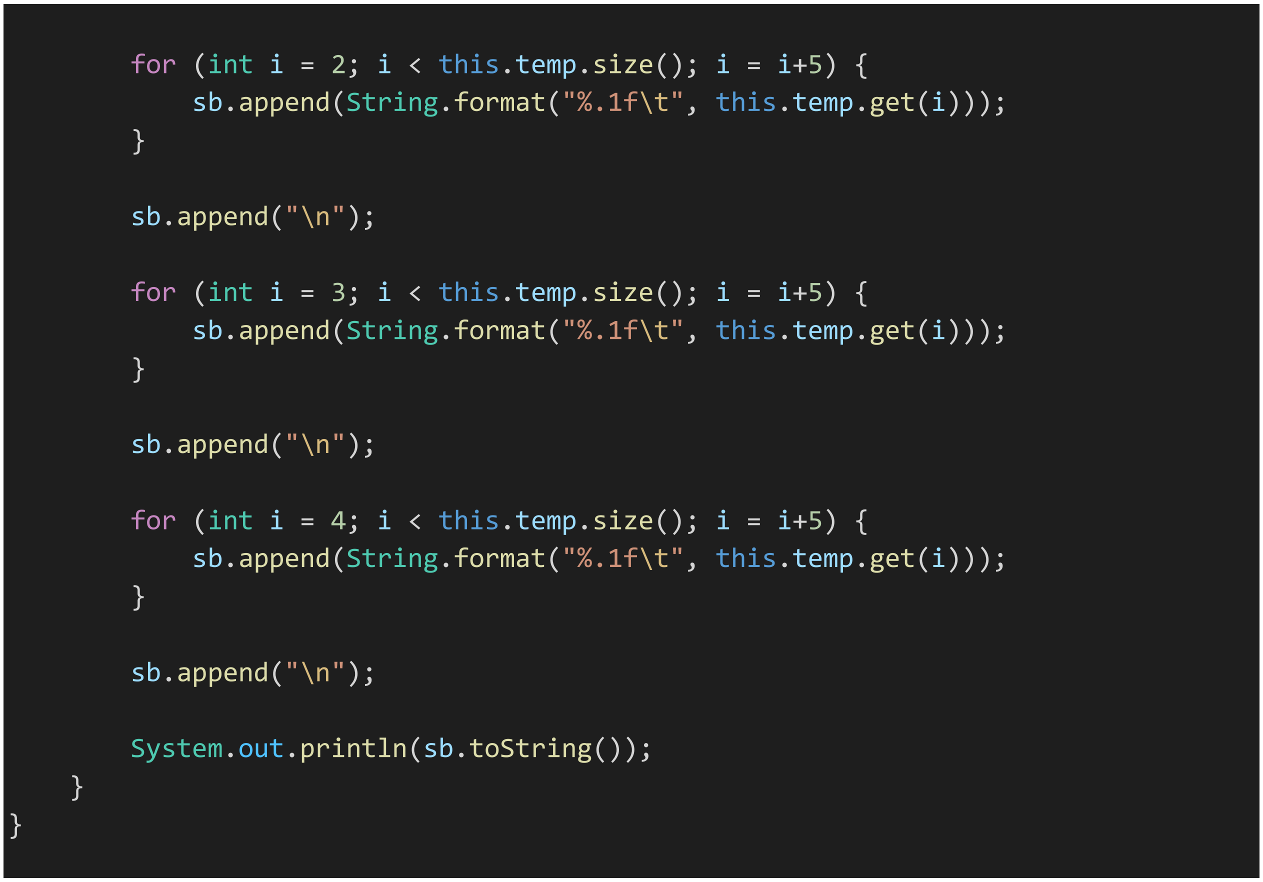
```java
/*
 * File Name:                 OneRow_Observer.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise E
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

import java.util.ArrayList;

public class OneRow_Observer implements Observer {
    private Subject subject;
    private ArrayList<Double> temp;

    OneRow_Observer(Subject s) {
        this.subject = s;
        this.subject.registerObserver(this);
    }

    @Override
    public void update(ArrayList<Double> arr) {
        System.out.println("\nNotification to One-Row Table Observer: Data Changed:");
        this.temp = arr;
        this.display();
    }

    public void display() {
        for (int i = 0; i < this.temp.size(); i++) {
            System.out.printf("%.1f ", this.temp.get(i));
        }

        System.out.println("\n");
    }
}
```

Exercise E – Source file ThreeColumnTable_Observer.java

```java
/*
 * File Name:                  ThreeColumnTable_Observer.java
 * Course:                     ENSF 614 - Fall 2021
 * Lab # and Assignment #:     Lab 6 Exercise E
 * Lab section:                B01
 * Completed by:               Bhavyai Gupta
 * Submission Date:            November 16, 2021
 */

import java.util.ArrayList;

public class ThreeColumnTable_Observer implements Observer {
    private Subject subject;
    private ArrayList<Double> temp;

    ThreeColumnTable_Observer(Subject s) {
        this.subject = s;
        this.subject.registerObserver(this);
    }

    @Override
    public void update(ArrayList<Double> arr) {
        System.out.printf("\nNotification to Three-Column Table Observer: Data Changed:");
        this.temp = arr;
        this.display();
    }

    public void display() {
        for (int i = 0; i < this.temp.size(); i++) {
            if (i % 3 == 0) {
                System.out.println();
            }

            System.out.printf("%.1f ", this.temp.get(i));
        }

        System.out.println("\n");
    }
}
```

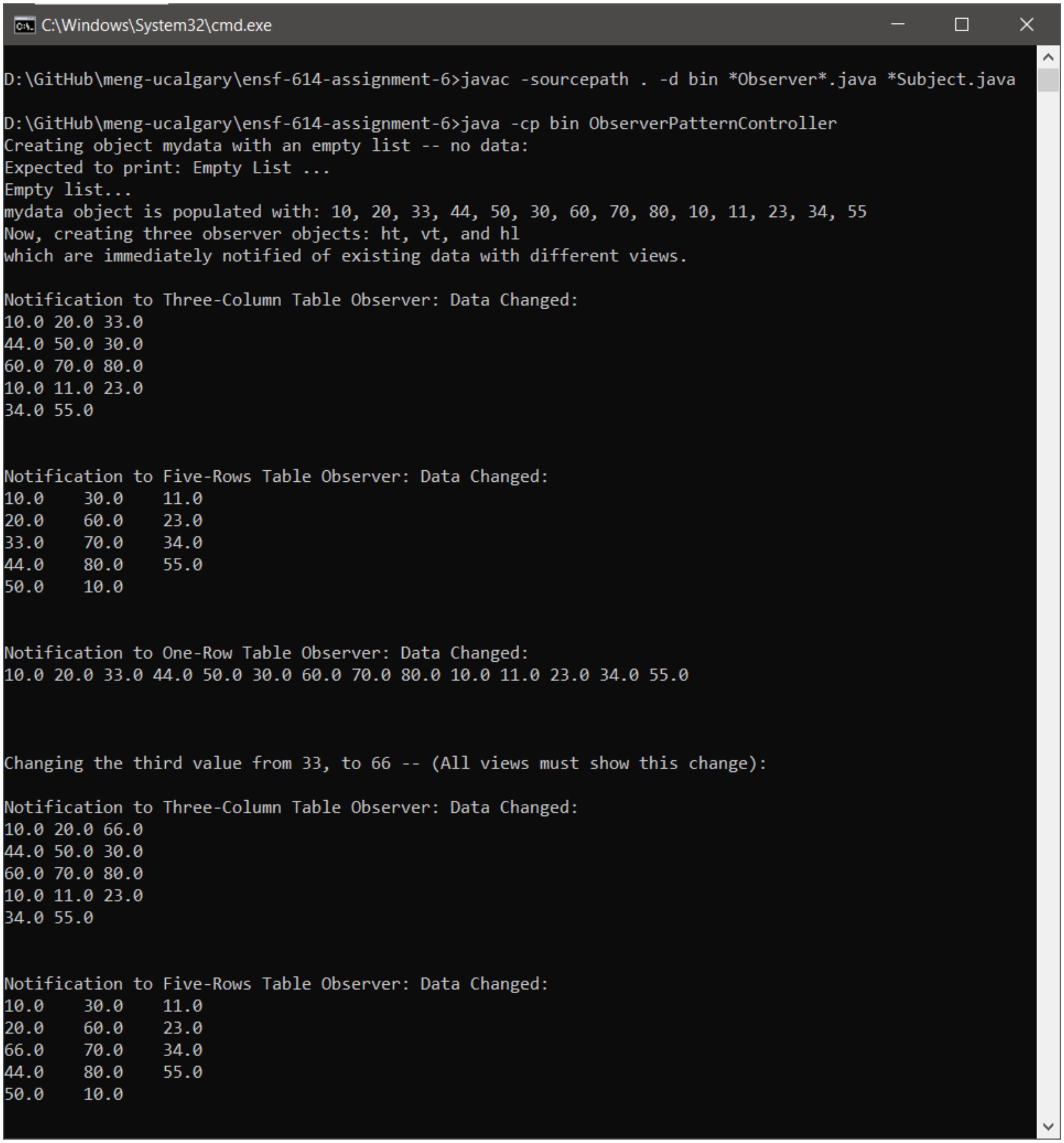
Exercise E – Source file FiveRowsTable_Observer.java

```
/*
 * File Name:                 FiveRowsTable_Observer.java
 * Course:                    ENSF 614 - Fall 2021
 * Lab # and Assignment #:    Lab 6 Exercise E
 * Lab section:               B01
 * Completed by:              Bhavyai Gupta
 * Submission Date:           November 16, 2021
 */

import java.util.ArrayList;

public class FiveRowsTable_Observer implements Observer {
    private Subject subject;
    private ArrayList<Double> temp;

    FiveRowsTable_Observer(Subject s) {
        this.subject = s;
        this.subject.registerObserver(this);
    }

    @Override
    public void update(ArrayList<Double> arr) {
        System.out.println("\nNotification to Five-Rows Table Observer: Data
Changed:");
        this.temp = arr;
        this.display();
    }

    public void display() {
        StringBuffer sb = new StringBuffer();

        for (int i = 0; i < this.temp.size(); i = i+5) {
            sb.append(String.format("%.1f\t", this.temp.get(i)));
        }

        sb.append("\n");

        for (int i = 1; i < this.temp.size(); i = i+5) {
            sb.append(String.format("%.1f\t", this.temp.get(i)));
        }

        sb.append("\n");
```

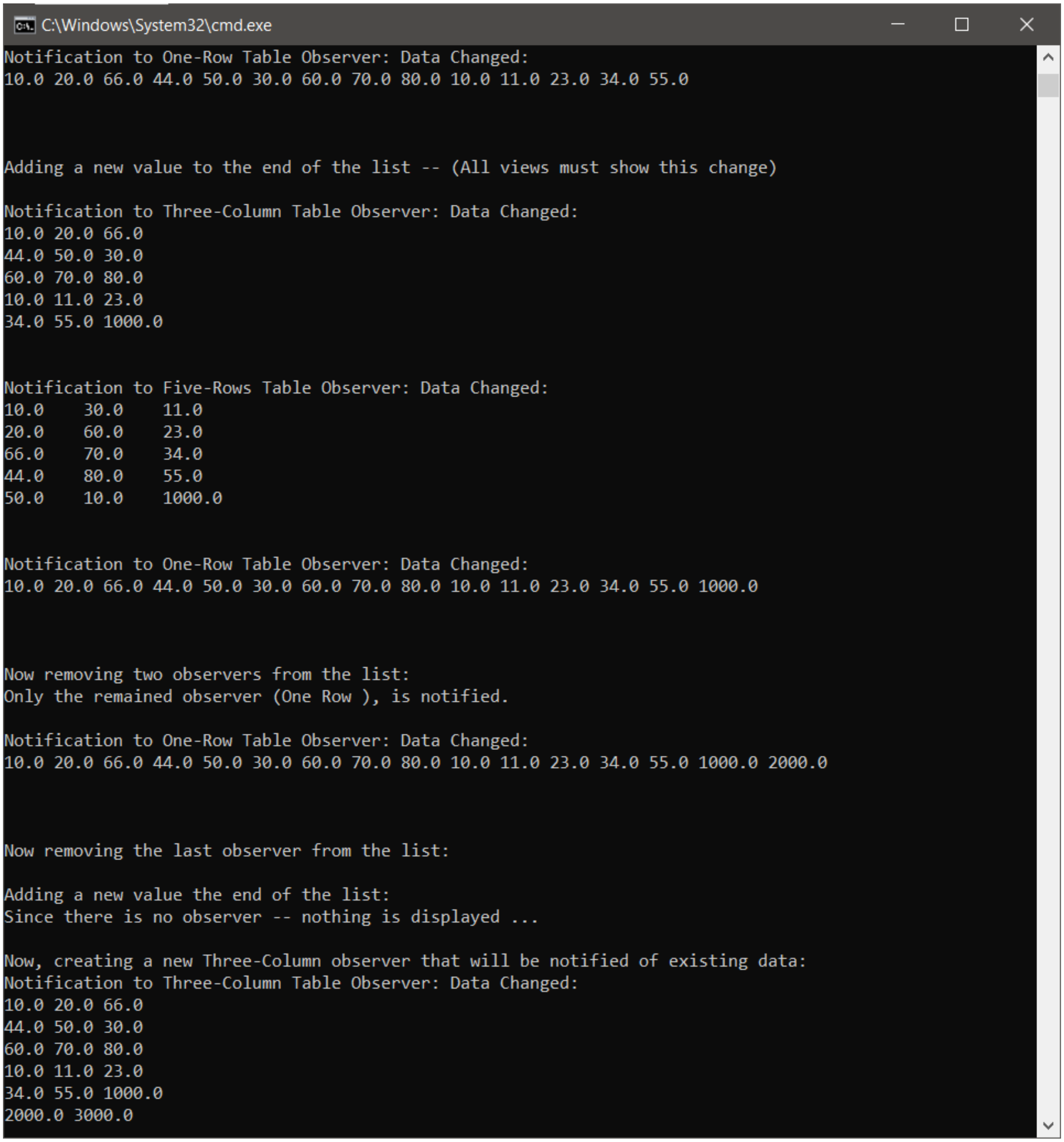


```
        for (int i = 2; i < this.temp.size(); i = i+5) {
            sb.append(String.format("%.1f\t", this.temp.get(i)));
        }

        sb.append("\n");

        for (int i = 3; i < this.temp.size(); i = i+5) {
            sb.append(String.format("%.1f\t", this.temp.get(i)));
        }

        sb.append("\n");

        for (int i = 4; i < this.temp.size(); i = i+5) {
            sb.append(String.format("%.1f\t", this.temp.get(i)));
        }

        sb.append("\n");

        System.out.println(sb.toString());
    }
}
```

Exercise E – Program Output



```
D:\GitHub\meng-ucalgary\ensf-614-assignment-6>javac -sourcepath . -d bin *Observer*.java *Subject.java

D:\GitHub\meng-ucalgary\ensf-614-assignment-6>java -cp bin ObserverPatternController
Creating object mydata with an empty list -- no data:
Expected to print: Empty List ...
Empty list...
mydata object is populated with: 10, 20, 33, 44, 50, 30, 60, 70, 80, 10, 11, 23, 34, 55
Now, creating three observer objects: ht, vt, and hl
which are immediately notified of existing data with different views.

Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 33.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0


Notification to Five-Rows Table Observer: Data Changed:
10.0    30.0    11.0
20.0    60.0    23.0
33.0    70.0    34.0
44.0    80.0    55.0
50.0    10.0


Notification to One-Row Table Observer: Data Changed:
10.0 20.0 33.0 44.0 50.0 30.0 60.0 70.0 80.0 10.0 11.0 23.0 34.0 55.0



Changing the third value from 33, to 66 -- (All views must show this change):

Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 66.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0


Notification to Five-Rows Table Observer: Data Changed:
10.0    30.0    11.0
20.0    60.0    23.0
66.0    70.0    34.0
44.0    80.0    55.0
50.0    10.0
```



```
Notification to One-Row Table Observer: Data Changed:
10.0 20.0 66.0 44.0 50.0 30.0 60.0 70.0 80.0 10.0 11.0 23.0 34.0 55.0



Adding a new value to the end of the list -- (All views must show this change)

Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 66.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0 1000.0


Notification to Five-Rows Table Observer: Data Changed:
10.0    30.0    11.0
20.0    60.0    23.0
66.0    70.0    34.0
44.0    80.0    55.0
50.0    10.0    1000.0


Notification to One-Row Table Observer: Data Changed:
10.0 20.0 66.0 44.0 50.0 30.0 60.0 70.0 80.0 10.0 11.0 23.0 34.0 55.0 1000.0



Now removing two observers from the list:
Only the remained observer (One Row ), is notified.

Notification to One-Row Table Observer: Data Changed:
10.0 20.0 66.0 44.0 50.0 30.0 60.0 70.0 80.0 10.0 11.0 23.0 34.0 55.0 1000.0 2000.0



Now removing the last observer from the list:

Adding a new value the end of the list:
Since there is no observer -- nothing is displayed ...

Now, creating a new Three-Column observer that will be notified of existing data:
Notification to Three-Column Table Observer: Data Changed:
10.0 20.0 66.0
44.0 50.0 30.0
60.0 70.0 80.0
10.0 11.0 23.0
34.0 55.0 1000.0
2000.0 3000.0
```