ENSF 694 – Summer 2024

Principles of Software Development II

University of Calgary

Lab Assignment 4

Student Name: Yael Gonzalez

Instructor: M. Moussavi, PhD, Peng

Submission Date: July 26, 2024

Exercise A

Part 1

Source code

lookupTable.h

```
/**
 *  File Name: lookupTable.h
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise A Part 1
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
 *  Submission Date: July 26, 2024
 */

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

// class LookupTable: GENERAL CONCEPTS
//
//    key/datum pairs are ordered.  The first pair is the pair with
//    the lowest key, the second pair is the pair with the second
//    lowest key, and so on.  This implies that you must be able to
//    compare two keys with the < operator.
//
//    Each LookupTable has an embedded iterator class that allows users
//    of the class to traverse trhough the list and  have acess to each
//    node.
//    In this version of the LookupTable a new struct type called Pair
//    is introduced which represents a key/data pair.

typedef string Type;

struct Pair
{
    int key;
    Type datum;

    // This ctor is writtent for convenience in creating objects of Pair and copy
    Pair(int keyA, Type datumA) : key(keyA), datum(datumA) {}
};

struct LT_Node
```

```
{
    Pair pairM;
    LT_Node *nextM;

    // This ctor should be convenient in insert and copy operations.
    LT_Node(const Pair &pairA, LT_Node *nextA) : pairM(pairA), nextM(nextA) {}
    // PROMISES: initializes the data members pairM and nextM, with pairA and
nextA
    // respectively
};

class LookupTable
{
public:
    // Nested class
    LookupTable() : sizeM(0), headM(nullptr), cursorM(nullptr) {}
    // PROMISES: An empty LookupTable object with all data members. Setting size,
cursor and
    // head to zero or nullptr
    // copy ctor
    LookupTable(const LookupTable &source);

    // assignment operator
    LookupTable &operator=(const LookupTable &rhs);

    // dtor
    ~LookupTable();

    LookupTable &begin();
    // PROMISES: Moves cursorM to the beginning of the list

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

    const Type &cursor_datum() const;
    // REQUIRES: cursor_ok()
```

```
    // PROMISES: Returns datum of key/datum pair to which cursor is attached.

    void insert(const Pair &pairA);
    // PROMISES:
    //   If keyA matches a key in the table, the datum for that
    //   key is set equal to datumA.
    //   If keyA does not match an existing key, keyA and datumM are
    //   used to create a new key/datum pair in the table.
    //   In either case, the cursor goes to the off-list state.

    int remove(const int &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 int &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.

    void display() const;
    // PROMISES: displays the values o
    bool isEmpty() const;
    // PROMISES: returns true is list is empty

    int *retrieve_at(int i);
```

```
    // PROMISES: returns the adress of the key at the position index. Reminder:
the index
    // number for the first node in the list is 0, 2nd node is 1, 3rd node is 2
and so on.

private:
    int sizeM;        // size of list (number of availble nodes)
    LT_Node *headM;   // pointer to the first node in the list
    LT_Node *cursorM; // pointer that can travers through the list

    void destroy();
    // Deallocate all nodes, set headM to zero.
    void copy(const LookupTable &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
```

lookupTable.cpp

```
/**
 *  File Name: lookupTable.cpp
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise A Part 1
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
 *  Submission Date: July 26, 2024
 */

#include "lookupTable.h"

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

LookupTable &LookupTable::operator=(const LookupTable &rhs)
{
    if (this != &rhs)
    {
        destroy();
        copy(rhs);
    }
```

```
    return *this;
}

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

LookupTable &LookupTable::begin()
{
    cursorM = headM;
    return *this;
}

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

int LookupTable::cursor_ok() const
{
    return cursorM != nullptr;
}

const int &LookupTable::cursor_key() const
{
    if (cursorM == nullptr)
    {
        cerr << "Cursor is pointing to null.";
    }

    return cursorM->pairM.key;
}

const Type &LookupTable::cursor_datum() const
{
    if (cursorM == nullptr)
    {
        cerr << "Cursor is pointing to null.";
    }

    return cursorM->pairM.datum;
}

void LookupTable::insert(const Pair &pairA)
```

```
{
    LT_Node *new_node = new LT_Node(pairA, nullptr);

    // Case 1: List is empty
    if (headM == nullptr)
    {
        headM = new_node;
        sizeM++;
    }
    // Case 2: First node's key is greater than new insert
    else if (pairA.key < headM->pairM.key)
    {
        new_node->nextM = headM;
        headM = new_node;
        sizeM++;
    }
    // Case 3: First node's key is equal to new insert
    else if (pairA.key == headM->pairM.key)
    {
        headM->pairM.datum = pairA.datum; // update datum only
        delete new_node;                  // delete unnecessary node
    }
    // Case 4: New insert to be added across the list (after first node)
    else
    {
        LT_Node *curr = headM;

        while (curr->nextM != nullptr && curr->nextM->pairM.key < pairA.key)
        {
            curr = curr->nextM;
        }

        if (curr->nextM != nullptr && curr->nextM->pairM.key == pairA.key)
        {
            curr->nextM->pairM.datum = pairA.datum; // update datum only
            delete new_node;                        // delete unnecessary node
        }
        else
        {
            new_node->nextM = curr->nextM;
            curr->nextM = new_node;
            sizeM++;
        }
    }
```

```
    cursorM = nullptr;
}

int LookupTable::remove(const int &keyA)
{
    // Case 1: List is empty
    if (headM == nullptr)
    {
        cerr << "List is empty." << endl;
        return 0;
    }

    LT_Node *curr = headM;
    LT_Node *prev = nullptr;
    int removed_key;

    // Case 2: Node to be removed is the first in the list
    if (headM->pairM.key == keyA)
    {
        headM = headM->nextM;
        removed_key = curr->pairM.key;
        delete curr;
        sizeM--;
        cursorM = nullptr;
        return removed_key;
    }

    // Case 3: Node to be removed is elsewhere in the list
    while (curr != nullptr && curr->pairM.key != keyA)
    {
        prev = curr;
        curr = curr->nextM;
    }

    if (curr == nullptr)
    {
        cerr << "Key not found." << endl;
        return 0;
    }

    prev->nextM = curr->nextM;
    removed_key = curr->pairM.key;
    delete curr;
    sizeM--;
    cursorM = nullptr;
```

```
    return removed_key;
}

void LookupTable::find(const int &keyA)
{
    for (LT_Node *curr = headM; curr != nullptr; curr = curr->nextM)
    {
        if (curr->pairM.key == keyA)
        {
            cursorM = curr;
            return;
        }
    }
    cursorM = nullptr;
}

void LookupTable::go_to_first()
{
    if (sizeM > 0)
    {
        cursorM = headM;
    }
}

void LookupTable::step_fwd()
{
    if (cursor_ok())
    {
        cursorM = cursorM->nextM;
    }
}

void LookupTable::make_empty()
{
    destroy();
    headM = nullptr;
    cursorM = nullptr;
    sizeM = 0;
}

void LookupTable::display() const
{
    if (headM == nullptr)
    {
        cerr << "List is empty.";
```

```
    }
    else
    {
        cout << " " << cursorM->pairM.key << " " << cursorM->pairM.datum << endl;
    }
}

bool LookupTable::isEmpty() const
{
    return headM == nullptr;
}

int *LookupTable::retrieve_at(int i)
{
    if (i < 0 || i >= sizeM)
    {
        cerr << "Index should be positive and less than " << sizeM << endl;
        return nullptr;
    }

    LT_Node *curr = headM;

    for (int j = 0; j < i; ++j)
    {
        curr = curr->nextM;
    }

    return &(curr->pairM.key);
}

void LookupTable::destroy()
{
    while (headM != nullptr)
    {
        LT_Node *temp = headM;
        headM = headM->nextM;
        delete temp;
    }
    cursorM = nullptr;
}

void LookupTable::copy(const LookupTable &source)
{
    if (source.headM == nullptr)
    {
```

```
        headM = nullptr;
        sizeM = 0;
        cursorM = nullptr;
        return;
    }

    headM = new LT_Node(source.headM->pairM, nullptr);
    LT_Node *srcNode = source.headM->nextM;
    LT_Node *thisNode = headM;

    while (srcNode != nullptr)
    {
        thisNode->nextM = new LT_Node(srcNode->pairM, nullptr);
        thisNode = thisNode->nextM;
        srcNode = srcNode->nextM;
    }

    sizeM = source.sizeM;

    if (source.cursorM != nullptr)
    {
        LT_Node *srcCursor = source.headM;
        LT_Node *newCursor = headM;

        while (srcCursor != source.cursorM)
        {
            srcCursor = srcCursor->nextM;
            newCursor = newCursor->nextM;
        }
        cursorM = newCursor;
    }
    else
    {
        cursorM = nullptr;
    }
}
```

Program output

```
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_A\part_1> g++ -Wall .\lockupTable.cpp .\lockupTable_tester-part1.cpp -o MyPart1_LockupTable
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_A\part_1> .\MyPart1_LockupTable.exe
Starting Test Run. Using input file.
Line 1 >> is comment
Line 2 >>  Passed
Line 3 >>  Passed
Line 4 >>  Passed
Line 5 >>  Passed
Line 6 >>  Passed
Line 7 >>  Passed
Line 8 >>  Passed
Line 9 >>  Passed
Line 10 >>  Passed
Line 11 >>  Passed
Line 12 >>  Passed
Line 13 >>  Passed
Line 14 >>  Passed
Line 15 >>  Passed
Line 16 >>  Passed
Line 17 >>  Passed
Line 18 >>  Passed
Line 19 >>  Passed
Line 20 >>  Passed
Line 21 >>  Passed
Reached End of Input File

MORE TESTS.....
Inserting 3 pairs:
Assert: three data must be in the list:
Okay. Passed.
Removing one pair with the key 8004:
Assert: one pair is removed.
Okay. Passed.

Printing table after inserting 3 and removing 1...

Expected to dispaly 8001 Tim Hardy and 8002 Joe Morrison:
 8001 Tim Hardy
 8002 Joe Morrison

Let's look up some keys 8001 and 8000...
Expected to find 8001 and NOT to find 8000...

Found key: 8001 Tim Hardy

Sorry, I couldn't find key: 8000 in the table.

Test copying: keys should be 8001, and 8002
 8001 Tim Hardy
 8002 Joe Morrison

Test assignment operator (key expected be 8001):
 8001 Tim Hardy

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

Program terminated successfully.

PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_A\part_1>
```

Part 2

Source code

Point.h

```
/**
 *  File Name: Point.h
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise A Part 2
 *  Created by: Mahmood Moussavi on 2024-05-10.
 *  Completed by: Yael Gonzalez
```

```
 *  Submission Date: July 26, 2024
 */

#ifndef Point_h
#define Point_h
#include <string.h>
class Point
{
public:
    Point(int x, int y, const char *label);
    ~Point();
    Point(const Point &src);
    Point &operator=(const Point &rhs);
    int getx() const;
    int gety() const;
    char *get_label() const;

private:
    int x, y;    // x and y coordinates of a point on Cartesian plain
    char *label; // pointer to an array allocated on the heap to store the label
for a point
};

#endif /* Point_h */
```

Point.cpp

```
/**
 *  File Name: Point.cpp
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise A Part 2
 *  Created by: Yael Gonzalez
 *  Submission Date: July 26, 2024
 */

#include "Point.h"

Point::Point(int x, int y, const char *label) : x(x), y(y), label(new
char[strlen(label + 1)])
{
    strcpy(this->label, label);
}

Point::~Point()
{
```

```
    delete[] label;
}

Point::Point(const Point &src) : x(src.x), y(src.y), label(new
char[strlen(src.label + 1)])
{
    strcpy(label, src.label);
}

Point &Point::operator=(const Point &rhs)
{
    if (this != &rhs) // avoid self copy
    {
        x = rhs.x;
        y = rhs.y;
        delete[] label;
        label = new char[strlen(rhs.label + 1)];
        strcpy(label, rhs.label);
    }
    return *this;
}

int Point::getx() const
{
    return x;
}

int Point::gety() const
{
    return y;
}

char *Point::get_label() const
{
    return label;
}
```

lookupTable.h

```
/**
 *  File Name: lookupTable.h
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise A Part 2
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
```

```
 *  Submission Date: July 26, 2024
 */

#ifndef LOOKUPTABLE_H
#define LOOKUPTABLE_H
#include "Point.h"
#include <iostream>
using namespace std;

// class LookupTable: GENERAL CONCEPTS
//
//    key/datum pairs are ordered.  The first pair is the pair with
//    the lowest key, the second pair is the pair with the second
//    lowest key, and so on.  This implies that you must be able to
//    compare two keys with the < operator.
//
//    Each LookupTable has an embedded iterator class that allows users
//    of the class to traverse trhough the list and  have acess to each
//    node.
//    In this version of the LookupTable a new struct type called Pair
//    is introduced which represents a key/data pair.

typedef Point Type;

struct Pair
{
    int key;
    Type datum;

    // This ctor is writtent for convenience in creating objects of Pair and copy
    Pair(int keyA, Type datumA) : key(keyA), datum(datumA) {}
};

struct LT_Node
{
    Pair pairM;
    LT_Node *nextM;

    // This ctor should be convenient in insert and copy operations.
    LT_Node(const Pair &pairA, LT_Node *nextA) : pairM(pairA), nextM(nextA) {}
    // PROMISES: initializes the data members pairM and nextM, with pairA and
nextA
    // respectively
};
```

```
class LookupTable
{
public:
    // Nested class
    LookupTable() : sizeM(0), headM(nullptr), cursorM(nullptr) {}
    // PROMISES: An empty LookupTable object with all data members. Setting size,
cursor and
    // head to zero or nullptr
    // copy ctor
    LookupTable(const LookupTable &source);

    // assignment operator
    LookupTable &operator=(const LookupTable &rhs);

    // dtor
    ~LookupTable();

    LookupTable &begin();
    // PROMISES: Moves cursorM to the beginning of the list

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

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

    void insert(const Pair &pairA);
    // PROMISES:
    //   If keyA matches a key in the table, the datum for that
    //   key is set equal to datumA.
    //   If keyA does not match an existing key, keyA and datumM are
    //   used to create a new key/datum pair in the table.
    //   In either case, the cursor goes to the off-list state.

    int remove(const int &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 int &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.

    void display() const;
    // PROMISES: displays the values o
    bool isEmpty() const;
    // PROMISES: returns true is list is empty

    int *retrieve_at(int i);
    // PROMISES: returns the adress of the key at the position index. Reminder:
the index
    // number for the first node in the list is 0, 2nd node is 1, 3rd node is 2
and so on.

private:
    int sizeM;        // size of list (number of availble nodes)
    LT_Node *headM;   // pointer to the first node in the list
    LT_Node *cursorM; // pointer that can travers through the list

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

```
    void copy(const LookupTable &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
```

lookupTable.cpp

```
/**
 *  File Name: lookupTable.cpp
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise A Part 2
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
 *  Submission Date: July 26, 2024
 */

#include "lookupTable.h"

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

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

    return *this;
}

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

LookupTable &LookupTable::begin()
{
    cursorM = headM;
    return *this;
```

```
}

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

int LookupTable::cursor_ok() const
{
    return cursorM != nullptr;
}

const int &LookupTable::cursor_key() const
{
    if (cursorM == nullptr)
    {
        cerr << "Cursor is pointing to null.";
    }

    return cursorM->pairM.key;
}

const Type &LookupTable::cursor_datum() const
{
    if (cursorM == nullptr)
    {
        cerr << "Cursor is pointing to null.";
    }

    return cursorM->pairM.datum;
}

void LookupTable::insert(const Pair &pairA)
{
    LT_Node *new_node = new LT_Node(pairA, nullptr);

    // Case 1: List is empty
    if (headM == nullptr)
    {
        headM = new_node;
        sizeM++;
    }
    // Case 2: First node's key is greater than new insert
    else if (pairA.key < headM->pairM.key)
    {
```

```
        new_node->nextM = headM;
        headM = new_node;
        sizeM++;
    }
    // Case 3: First node's key is equal to new insert
    else if (pairA.key == headM->pairM.key)
    {
        headM->pairM.datum = pairA.datum; // update datum only
        delete new_node;                  // delete unnecessary node
    }
    // Case 4: New insert to be added across the list (after first node)
    else
    {
        LT_Node *curr = headM;

        while (curr->nextM != nullptr && curr->nextM->pairM.key < pairA.key)
        {
            curr = curr->nextM;
        }

        if (curr->nextM != nullptr && curr->nextM->pairM.key == pairA.key)
        {
            curr->nextM->pairM.datum = pairA.datum; // update datum only
            delete new_node;                        // delete unnecessary node
        }
        else
        {
            new_node->nextM = curr->nextM;
            curr->nextM = new_node;
            sizeM++;
        }
    }

    cursorM = nullptr;
}

int LookupTable::remove(const int &keyA)
{
    // Case 1: List is empty
    if (headM == nullptr)
    {
        cerr << "List is empty." << endl;
        return 0;
    }
```

```
    LT_Node *curr = headM;
    LT_Node *prev = nullptr;
    int removed_key;

    // Case 2: Node to be removed is the first in the list
    if (headM->pairM.key == keyA)
    {
        headM = headM->nextM;
        removed_key = curr->pairM.key;
        delete curr;
        sizeM--;
        cursorM = nullptr;
        return removed_key;
    }

    // Case 3: Node to be removed is elsewhere in the list
    while (curr != nullptr && curr->pairM.key != keyA)
    {
        prev = curr;
        curr = curr->nextM;
    }

    if (curr == nullptr)
    {
        cerr << "Key not found." << endl;
        return 0;
    }

    prev->nextM = curr->nextM;
    removed_key = curr->pairM.key;
    delete curr;
    sizeM--;
    cursorM = nullptr;
    return removed_key;
}

void LookupTable::find(const int &keyA)
{
    for (LT_Node *curr = headM; curr != nullptr; curr = curr->nextM)
    {
        if (curr->pairM.key == keyA)
        {
            cursorM = curr;
            return;
        }
```

```
    }
    cursorM = nullptr;
}

void LookupTable::go_to_first()
{
    if (sizeM > 0)
    {
        cursorM = headM;
    }
}

void LookupTable::step_fwd()
{
    if (cursor_ok())
    {
        cursorM = cursorM->nextM;
    }
}

void LookupTable::make_empty()
{
    destroy();
    headM = nullptr;
    cursorM = nullptr;
    sizeM = 0;
}

void LookupTable::display() const
{
    if (headM == nullptr)
    {
        cerr << "List is empty.";
    }
    else
    {
        cout << " " << cursorM->pairM.key << " " << cursorM->pairM.datum.getx()
<< ", " << cursorM->pairM.datum.gety() << ", " << cursorM-
>pairM.datum.get_label() << endl;
    }
}

bool LookupTable::isEmpty() const
{
    return headM == nullptr;
```

```
}

int *LookupTable::retrieve_at(int i)
{
    if (i < 0 || i >= sizeM)
    {
        cerr << "Index should be positive and less than " << sizeM << endl;
        return nullptr;
    }

    LT_Node *curr = headM;

    for (int j = 0; j < i; ++j)
    {
        curr = curr->nextM;
    }

    return &(curr->pairM.key);
}

void LookupTable::destroy()
{
    while (headM != nullptr)
    {
        LT_Node *temp = headM;
        headM = headM->nextM;
        delete temp;
    }
    cursorM = nullptr;
}

void LookupTable::copy(const LookupTable &source)
{
    if (source.headM == nullptr)
    {
        headM = nullptr;
        sizeM = 0;
        cursorM = nullptr;
        return;
    }

    headM = new LT_Node(source.headM->pairM, nullptr);
    LT_Node *srcNode = source.headM->nextM;
    LT_Node *thisNode = headM;
```

```
    while (srcNode != nullptr)
    {
        thisNode->nextM = new LT_Node(srcNode->pairM, nullptr);
        thisNode = thisNode->nextM;
        srcNode = srcNode->nextM;
    }

    sizeM = source.sizeM;

    if (source.cursorM != nullptr)
    {
        LT_Node *srcCursor = source.headM;
        LT_Node *newCursor = headM;

        while (srcCursor != source.cursorM)
        {
            srcCursor = srcCursor->nextM;
            newCursor = newCursor->nextM;
        }
        cursorM = newCursor;
    }
    else
    {
        cursorM = nullptr;
    }
}
```

Program output

```
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_A\part_2> g++ -Wall .\lookupTable.cpp .\lookupTable_tester_part2.cpp .\Point.cpp -o MyPart2_Lo
okupTable
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_A\part_2> .\MyPart2_LookupTable.exe
Starting Test Run. Using input file.
Line 1 >> is comment
Line 2 >>  Passed
Line 3 >>  Passed
Line 4 >>  Passed
Line 5 >>  Passed
Line 6 >>  Passed
Line 7 >>  Passed
Line 8 >>  Passed
Line 9 >>  Passed
Line 10 >>  Passed
Line 11 >>  Passed
Line 12 >>  Passed
Line 13 >>  Passed
Line 14 >>  Passed
Line 15 >>  Passed
Line 16 >>  Passed
Line 17 >>  Passed
Line 18 >>  Passed
Line 19 >>  Passed
Line 20 >>  Passed
Line 21 >>  Passed
Exiting...
Finishing Test Run
Showing Data in the List:


Program terminated successfully.

PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_A\part_2>
```

Exercise B

Source code

```cpp
/**
 *  File Name: lab4exe_B.cpp
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise B
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
 *  Submission Date: July 26, 2024
 */

void print_from_binary(char *filename)
{
    ifstream is(filename, ios::binary);
    if (is.fail())
    {
        cerr << "failed to open file: " << filename << endl;
        exit(1);
    }

    City cities[size];

    for (int i = 0; i < size; i++)
        is.read((char *)(&cities[i]), sizeof(City));

    for (int i = 0; i < size; i++)
```

```
        cout << "Name: " << cities[i].name << ", x coordinate: " << cities[i].x
<< ", y coordinate: " << cities[i].y << endl;

    is.close();
}
```

Program output

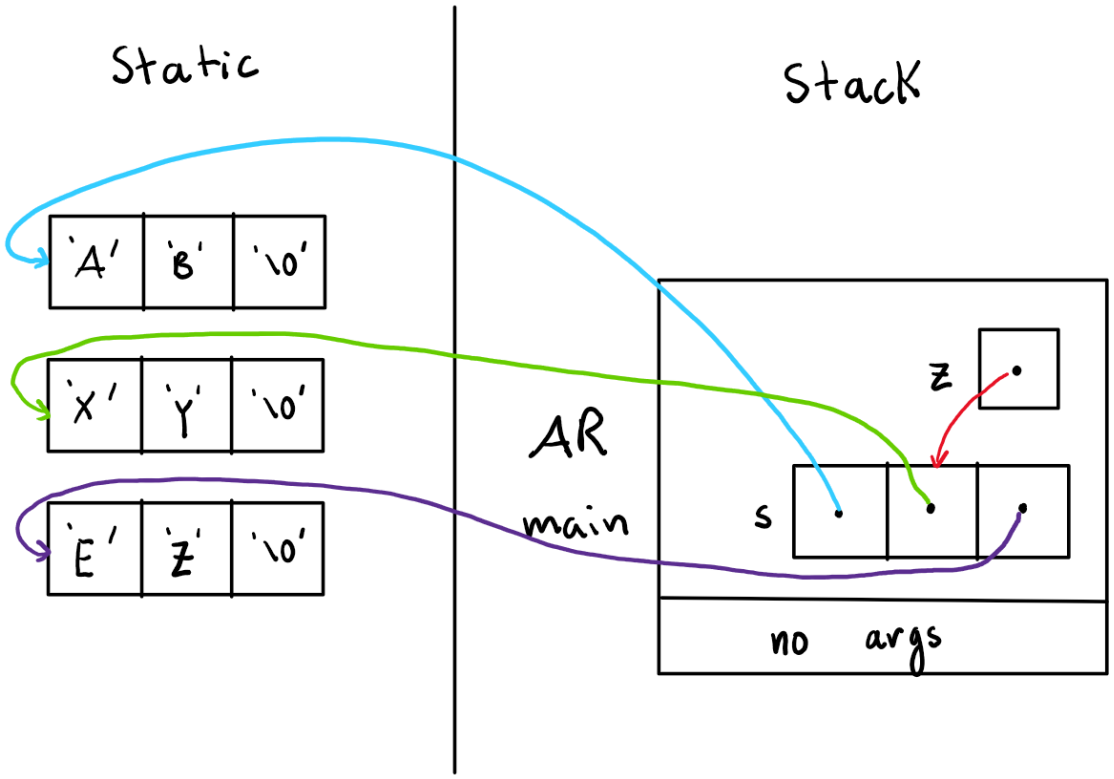
```
PROBLEMS 12  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE  GITLENS

PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_B> g++ -std=gnu++11 -Wall .\lab4exe_B.cpp -o myProgram
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_B> .\myProgram.exe

The content of the binary file is:
Name: Calgary, x coordinate: 100, y coordinate: 50
Name: Edmonton, x coordinate: 100, y coordinate: 150
Name: Vancouver, x coordinate: 50, y coordinate: 50
Name: Regina, x coordinate: 200, y coordinate: 50
Name: Toronto, x coordinate: 500, y coordinate: 50
Name: Montreal, x coordinate: 200, y coordinate: 50
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_B>
```

Exercise C

AR Diagram



Source code

```
/**
 *  File Name: lab4exe_C.cpp
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise C
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
 *  Submission Date: July 26, 2024
 */

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

void insertion_sort(int *int_array, int n);
/* REQUIRES
 *    n > 0.
 *    Array elements int_array[0] ... int_array[n - 1] exist.
 * PROMISES
 *    Element values are rearranged in non-decreasing order.
 */

void insertion_sort(const char **str_array, int n);

/* REQUIRES
 *   n > 0.
 *   Array elements str_array[0] ... str_array[n - 1] exist.
 * PROMISES
 *   pointers in str_array are rearranged so that strings:
 *   str_array[0] points to a string with the smallest string (lexicographicall)
,
 *   str_array[1] points to the second smallest string, ..., str_array[n-2]
 *   points to the second largest, and str_array[n-1] points to the largest
string
 */

int main(void)
{
    const char *s[] = {"AB", "XY", "EZ"};
    const char **z = s;
    z += 1;

    cout << "The value of **z is: " << **z << endl;
    cout << "The value of *z is: " << *z << endl;
    cout << "The value of **(z-1) is: " << **(z - 1) << endl;
```

```
    cout << "The value of *(z-1) is: " << *(z - 1) << endl;
    cout << "The value of z[1][1] is: " << z[1][1] << endl;
    cout << "The value of *(*(z+1)+1) is: " << *(*(z + 1) + 1) << endl;

    // point 1

    int a[] = {413, 282, 660, 171, 308, 537};

    int i;
    int n_elements = sizeof(a) / sizeof(int);

    cout << "Here is your array of integers before sorting: \n";
    for (i = 0; i < n_elements; i++)
        cout << a[i] << endl;
    cout << endl;

    insertion_sort(a, n_elements);

    cout << "Here is your array of ints after sorting:  \n";
    for (i = 0; i < n_elements; i++)
        cout << a[i] << endl;
#if 1
    const char *strings[] = {"Red", "Blue", "pink", "apple", "almond", "white",
                             "nut", "Law", "cup"};

    n_elements = sizeof(strings) / sizeof(char *);

    cout << "\nHere is your array of strings before sorting: \n";
    for (i = 0; i < n_elements; i++)
        cout << strings[i] << endl;
    cout << endl;

    insertion_sort(strings, 9);

    cout << "Here is your array of strings after sorting:  \n";
    for (i = 0; i < n_elements; i++)
        cout << strings[i] << endl;
    cout << endl;

#endif

    return 0;
}

void insertion_sort(int *a, int n)
```

```c
{
    int i;
    int j;
    int value_to_insert;

    for (i = 1; i < n; i++)
    {
        value_to_insert = a[i];

        /* Shift values greater than value_to_insert. */
        j = i;
        while (j > 0 && a[j - 1] > value_to_insert)
        {
            a[j] = a[j - 1];
            j--;
        }

        a[j] = value_to_insert;
    }
}

void insertion_sort(const char **str_array, int n)
{
    int i;
    int j;
    const char *str_to_insert;

    for (i = 1; i < n; i++)
    {
        str_to_insert = *(str_array + i);

        /* Shift strings lexicographically greater than str_to_insert. */
        j = i;
        while (j > 0 && strcmp(*(str_array + j - 1), str_to_insert) > 0)
        {
            *(str_array + j) = str_array[j - 1];
            j--;
        }

        *(str_array + j) = str_to_insert;
    }
}
```

Program output

```
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_C> g++ -Wall .\lab4exe_C.cpp -o MyProgram
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_C> .\MyProgram.exe
The value of **z is: X
The value of *z is: XY
The value of **(z-1) is: A
The value of *(z-1) is: AB
The value of z[1][1] is: Z
The value of *(*(z+1)+1) is: Z
Here is your array of integers before sorting:
413
282
660
171
308
537

Here is your array of ints after sorting:
171
282
308
413
537
660

Here is your array of strings before sorting:
Red
Blue
pink
apple
almond
white
nut
Law
cup

Here is your array of strings after sorting:
Blue
Law
Red
almond
apple
cup
nut
pink
white

PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_C>
```

Exercise D

Source code

```
/**
 *  File Name: matrix.cpp
 *  Assignment: ENSF 694 Summer 2024 - Lab 4 Exercise D
 *  Created by: Mahmood Moussavi
 *  Completed by: Yael Gonzalez
```

```
 *  Submission Date: July 26, 2024
 */

#include "matrix.h"

Matrix::Matrix(int r, int c) : rowsM(r), colsM(c)
{
    matrixM = new double *[rowsM];
    assert(matrixM != NULL);

    for (int i = 0; i < rowsM; i++)
    {
        matrixM[i] = new double[colsM];
        assert(matrixM[i] != NULL);
    }
    sum_rowsM = new double[rowsM];
    assert(sum_rowsM != NULL);

    sum_colsM = new double[colsM];
    assert(sum_colsM != NULL);
}

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

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

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

    return *this;
}

double Matrix::get_sum_col(int i) const
{
```

```
    assert(i >= 0 && i < colsM);
    return sum_colsM[i];
}

double Matrix::get_sum_row(int i) const
{
    assert(i >= 0 && i < rowsM);
    return sum_rowsM[i];
}

void Matrix::sum_of_rows() const
{
    double sum;
    for (int i = 0; i < rowsM; i++)
    {
        sum = 0.0;
        for (int j = 0; j < colsM; j++)
        {
            sum += matrixM[i][j];
        }
        sum_rowsM[i] = sum;
    }
}

void Matrix::sum_of_cols() const
{
    double sum;
    for (int j = 0; j < colsM; j++)
    {
        sum = 0.0;
        for (int i = 0; i < rowsM; i++)
        {
            sum += matrixM[i][j];
        }
        sum_colsM[j] = sum;
    }
}

void Matrix::copy(const Matrix &source)
{
    if (source.matrixM == NULL)
    {
        matrixM = NULL;
        sum_rowsM = NULL;
        sum_colsM = NULL;
```

```
        rowsM = 0;
        colsM = 0;
        return;
    }

    rowsM = source.rowsM;
    colsM = source.colsM;

    sum_rowsM = new double[rowsM];
    assert(sum_rowsM != NULL);
    for (int i = 0; i < rowsM; i++)
        sum_rowsM[i] = source.sum_rowsM[i];

    sum_colsM = new double[colsM];
    assert(sum_colsM != NULL);
    for (int i = 0; i < colsM; i++)
        sum_colsM[i] = source.sum_colsM[i];

    matrixM = new double *[rowsM];
    assert(matrixM != NULL);
    for (int i = 0; i < rowsM; i++)
    {
        matrixM[i] = new double[colsM];
        assert(matrixM[i] != NULL);
        for (int j = 0; j < colsM; j++)
        {
            matrixM[i][j] = source.matrixM[i][j];
        }
    }
}

void Matrix::destroy()
{
    if (matrixM != NULL)
    {
        for (int i = 0; i < rowsM; i++)
        {
            delete[] matrixM[i];
        }
        delete[] matrixM;
    }
    delete[] sum_rowsM;
    delete[] sum_colsM;
}
```

Program output

```
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_D> g++ -Wall .\matrix.cpp .\lab4exe_D.cpp -o myMatrix
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_D> .\myMatrix.exe 3 4


The values in matrix m1 are:

   2.3   3.0   3.7   4.3
   2.7   3.3   4.0   4.7
   3.0   3.7   4.3   5.0


The values in matrix m2 are:
   2.7   3.3   4.0   4.7   5.3   6.0
   3.0   3.7   4.3   5.0   5.7   6.3
   3.3   4.0   4.7   5.3   6.0   6.7
   3.7   4.3   5.0   5.7   6.3   7.0


The new values in matrix m1 and sum of its rows and columns are
   2.7   3.3   4.0   4.7   5.3   6.0 | 26.0
   3.0   3.7   4.3   5.0   5.7   6.3 | 28.0
   3.3   4.0   4.7   5.3   6.0   6.7 | 30.0
   3.7   4.3   5.0   5.7   6.3   7.0 | 32.0
     -----------------------------------
  12.7  15.3  18.0  20.7  23.3  26.0


The values in matrix m3 and sum of its rows and columns are:
   5.0   3.3   4.0   4.7   5.3   6.0 | 28.3
   3.0  15.0   4.3   5.0   5.7   6.3 | 39.3
   3.3   4.0  25.0   5.3   6.0   6.7 | 50.3
   3.7   4.3   5.0   5.7   6.3   7.0 | 32.0
     -----------------------------------
  15.0  26.7  38.3  20.7  23.3  26.0

The new values in matrix m2 are:
  -5.0   3.3   4.0   4.7   5.3   6.0 | 18.3
   3.0 -15.0   4.3   5.0   5.7   6.3 |  9.3
   3.3   4.0 -25.0   5.3   6.0   6.7 |  0.3
   3.7   4.3   5.0   5.7   6.3   7.0 | 32.0
     -----------------------------------
   5.0  -3.3 -11.7  20.7  23.3  26.0


The values in matrix m3 and sum of it rows and columns are still the same:
   5.0   3.3   4.0   4.7   5.3   6.0 | 28.3
   3.0  15.0   4.3   5.0   5.7   6.3 | 39.3
   3.3   4.0  25.0   5.3   6.0   6.7 | 50.3
   3.7   4.3   5.0   5.7   6.3   7.0 | 32.0
     -----------------------------------
  15.0  26.7  38.3  20.7  23.3  26.0
PS C:\Users\Owner\Desktop\Calgary\ENSF694\assignments\a4-ensf694\ex_D>
```