Course: ENSF694 – Summer 2025

Lab #: Lab 4

Instructor: Mahmood Moussavi

Student Name: John Zhou

Submission Date: August 1, 2025  
  
  
  
  
  
  
I have been keeping all the files in github. I hope by providing this github link will help you a little bit.  
https://github.com/JZ-Zhou-UofC/ENSF-604-assignment-repo

Exercise A Part I

Code:  
/\*

 \* lookupTable.cpp

 \*  ENSF 694 Lab 4, exercise A part I

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#include "lookupTable.h"

LT\_Node::LT\_Node(const Pair &pairA, LT\_Node \*nextA) : pairM(pairA), nextM(nextA)

{

}

LookupTable::LookupTable() : sizeM(0), headM(nullptr), cursorM(nullptr)

{

}

LookupTable::LookupTable(const LookupTable &source) : headM(nullptr), cursorM(nullptr), sizeM(source.sizeM)

{

    if (source.headM == nullptr)

    {

        return;

    }

    headM = new LT\_Node(source.headM->pairM, nullptr);

    LT\_Node \*sourcePtr = source.headM->nextM;

    LT\_Node \*destPtr = headM;

    if (source.cursorM == source.headM)

    {

        cursorM = headM;

    }

    while (sourcePtr != nullptr)

    {

        destPtr->nextM = new LT\_Node(sourcePtr->pairM, nullptr);

        if (source.cursorM == sourcePtr)

        {

            cursorM = destPtr->nextM;

        }

        destPtr = destPtr->nextM;

        sourcePtr = sourcePtr->nextM;

    }

}

LookupTable &LookupTable::operator=(const LookupTable &rhs)

{

    if (this == &rhs)

    {

        return \*this;

    }

    LT\_Node \*current = headM;

    while (current != nullptr)

    {

        LT\_Node \*temp = current;

        current = current->nextM;

        delete temp;

    }

    headM = new LT\_Node(rhs.headM->pairM, nullptr);

    LT\_Node \*rhsPtr = rhs.headM->nextM;

    LT\_Node \*destPtr = headM;

    if (rhs.cursorM == rhs.headM)

    {

        cursorM = headM;

    }

    while (rhsPtr != nullptr)

    {

        destPtr->nextM = new LT\_Node(rhsPtr->pairM, nullptr);

        if (rhs.cursorM == rhsPtr)

        {

            cursorM = destPtr->nextM;

        }

        destPtr = destPtr->nextM;

        rhsPtr = rhsPtr->nextM;

    }

    sizeM = rhs.sizeM;

    return \*this;

}

LookupTable::~LookupTable()

{

    LT\_Node \*current = headM;

    while (current != nullptr)

    {

        LT\_Node \*temp = current;

        current = current->nextM;

        delete temp;

    }

}

LookupTable &LookupTable::begin()

{

    cursorM = headM;

    return \*this;

}

int LookupTable::size() const

{

    return sizeM;

}

int LookupTable::cursor\_ok() const

{

    return cursorM == nullptr ? 0 : 1;

}

const int &LookupTable::cursor\_key() const

{

    return cursorM->pairM.key;

}

const Type &LookupTable::cursor\_datum() const

{

    return cursorM->pairM.datum;

}

void LookupTable::insert(const Pair &pairA)

{

    LT\_Node \*current = headM;

    cursorM = nullptr;

    while (current != nullptr)

    {

        if (current->pairM.key == pairA.key)

        {

            current->pairM.datum = pairA.datum;

            return;

        }

        current = current->nextM;

    }

    LT\_Node \*newElement = new LT\_Node(pairA, nullptr);

    if (headM == nullptr || pairA.key < headM->pairM.key)

    {

        newElement->nextM = headM;

        headM = newElement;

    }

    else

    {

        LT\_Node \*prevNode = headM;

        while (prevNode->nextM != nullptr && prevNode->nextM->pairM.key < pairA.key)

        {

            prevNode = prevNode->nextM;

        }

        newElement->nextM = prevNode->nextM;

        prevNode->nextM = newElement;

    }

    sizeM++;

}

int LookupTable::remove(const int &keyA)

{

    LT\_Node \*prev = nullptr;

    LT\_Node \*current = headM;

    while (current != nullptr)

    {

        if (current->pairM.key == keyA)

        {

            if (prev == nullptr)

            {

                headM = current->nextM;

            }

            else

            {

                prev->nextM = current->nextM;

            }

            delete current;

            sizeM--;

            cursorM = nullptr;

            return keyA;

        }

        prev = current;

        current = current->nextM;

    }

    cursorM = nullptr;

    return 0;

}

void LookupTable::find(const int &keyA)

{

    LT\_Node \*current = headM;

    while (current != nullptr)

    {

        if (current->pairM.key == keyA)

        {

            cursorM = current;

            return;

        }

        current = current->nextM;

    }

    cursorM = nullptr;

}

void LookupTable::go\_to\_first()

{

    if (sizeM > 0)

    {

        cursorM = headM;

    }

    else

    {

        cursorM = nullptr;

    }

}

void LookupTable::step\_fwd()

{

    if (!cursor\_ok())

    {

        return;

    }

    if (cursorM->nextM != nullptr)

    {

        cursorM = cursorM->nextM;

    }

    else

    {

        cursorM = nullptr;

    }

}

void LookupTable::make\_empty()

{

    LT\_Node \*current = headM;

    while (current != nullptr)

    {

        LT\_Node \*temp = current;

        current = current->nextM;

        delete temp;

    }

    headM = nullptr;

    cursorM = nullptr;

    sizeM = 0;

}

void LookupTable::display() const

{

    LT\_Node \*current = headM;

    while (current != nullptr)

    {

        std::cout << current->pairM.key << "   " << current->pairM.datum << endl;

        current = current->nextM;

    }

    std::cout << std::endl;

}

bool LookupTable::isEmpty() const

{

    return sizeM == 0;

}

int \*LookupTable::retrieve\_at(int i)

{

    if (i < 0 || i >= sizeM)

    {

        return nullptr;

    }

    LT\_Node \*current = headM;

    int count = 0;

    while (current != nullptr && count < i)

    {

        current = current->nextM;

        count++;

    }

    if (current != nullptr)

    {

        return &(current->pairM.key);

    }

    else

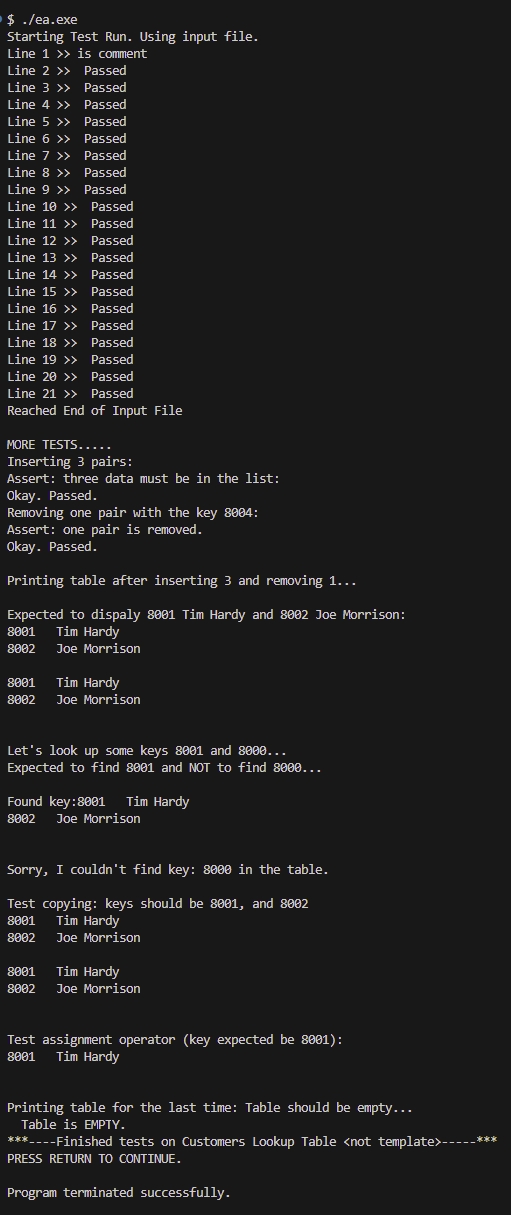
    {

        return nullptr;

    }

}

Execution output



Exercise A Part II

#include "lookupTable.h"

/\*

\* lookupTable.cpp

\* ENSF 694 Lab 4, exercise A part II

\* Created by Mahmood Moussavi

\* Completed by: John Zhou

\*/

LT\_Node::LT\_Node(const Pair &pairA, LT\_Node \*nextA) : pairM(pairA), nextM(nextA)

{

}

LookupTable::LookupTable() : sizeM(0), headM(nullptr), cursorM(nullptr)

{

}

LookupTable::LookupTable(const LookupTable &source) : headM(nullptr), cursorM(nullptr), sizeM(source.sizeM)

{

if (source.headM == nullptr)

{

return;

}

headM = new LT\_Node(source.headM->pairM, nullptr);

LT\_Node \*sourcePtr = source.headM->nextM;

LT\_Node \*destPtr = headM;

if (source.cursorM == source.headM)

{

cursorM = headM;

}

while (sourcePtr != nullptr)

{

destPtr->nextM = new LT\_Node(sourcePtr->pairM, nullptr);

if (source.cursorM == sourcePtr)

{

cursorM = destPtr->nextM;

}

destPtr = destPtr->nextM;

sourcePtr = sourcePtr->nextM;

}

}

LookupTable &LookupTable::operator=(const LookupTable &rhs)

{

if (this == &rhs)

{

return \*this;

}

LT\_Node \*current = headM;

while (current != nullptr)

{

LT\_Node \*temp = current;

current = current->nextM;

delete temp;

}

headM = new LT\_Node(rhs.headM->pairM, nullptr);

LT\_Node \*rhsPtr = rhs.headM->nextM;

LT\_Node \*destPtr = headM;

if (rhs.cursorM == rhs.headM)

{

cursorM = headM;

}

while (rhsPtr != nullptr)

{

destPtr->nextM = new LT\_Node(rhsPtr->pairM, nullptr);

if (rhs.cursorM == rhsPtr)

{

cursorM = destPtr->nextM;

}

destPtr = destPtr->nextM;

rhsPtr = rhsPtr->nextM;

}

sizeM = rhs.sizeM;

return \*this;

}

LookupTable::~LookupTable()

{

LT\_Node \*current = headM;

while (current != nullptr)

{

LT\_Node \*temp = current;

current = current->nextM;

delete temp;

}

}

LookupTable &LookupTable::begin()

{

cursorM = headM;

return \*this;

}

int LookupTable::size() const

{

return sizeM;

}

int LookupTable::cursor\_ok() const

{

return cursorM == nullptr ? 0 : 1;

}

const int &LookupTable::cursor\_key() const

{

return cursorM->pairM.key;

}

const Type &LookupTable::cursor\_datum() const

{

return cursorM->pairM.datum;

}

void LookupTable::insert(const Pair &pairA)

{

LT\_Node \*current = headM;

cursorM = nullptr;

while (current != nullptr)

{

if (current->pairM.key == pairA.key)

{

current->pairM.datum = pairA.datum;

return;

}

current = current->nextM;

}

LT\_Node \*newElement = new LT\_Node(pairA, nullptr);

if (headM == nullptr || pairA.key < headM->pairM.key)

{

newElement->nextM = headM;

headM = newElement;

}

else

{

LT\_Node \*prevNode = headM;

while (prevNode->nextM != nullptr && prevNode->nextM->pairM.key < pairA.key)

{

prevNode = prevNode->nextM;

}

newElement->nextM = prevNode->nextM;

prevNode->nextM = newElement;

}

sizeM++;

}

int LookupTable::remove(const int &keyA)

{

LT\_Node \*prev = nullptr;

LT\_Node \*current = headM;

while (current != nullptr)

{

if (current->pairM.key == keyA)

{

if (prev == nullptr)

{

headM = current->nextM;

}

else

{

prev->nextM = current->nextM;

}

delete current;

sizeM--;

cursorM = nullptr;

return keyA;

}

prev = current;

current = current->nextM;

}

cursorM = nullptr;

return 0;

}

void LookupTable::find(const int &keyA)

{

LT\_Node \*current = headM;

while (current != nullptr)

{

if (current->pairM.key == keyA)

{

cursorM = current;

return;

}

current = current->nextM;

}

cursorM = nullptr;

}

void LookupTable::go\_to\_first()

{

if (sizeM > 0)

{

cursorM = headM;

}

else

{

cursorM = nullptr;

}

}

void LookupTable::step\_fwd()

{

if (!cursor\_ok())

{

return;

}

if (cursorM->nextM != nullptr)

{

cursorM = cursorM->nextM;

}

else

{

cursorM = nullptr;

}

}

void LookupTable::make\_empty()

{

LT\_Node \*current = headM;

while (current != nullptr)

{

LT\_Node \*temp = current;

current = current->nextM;

delete temp;

}

headM = nullptr;

cursorM = nullptr;

sizeM = 0;

}

void LookupTable::display() const

{

LT\_Node \*current = headM;

while (current != nullptr)

{

std::cout << current->pairM.key << "x = " << current->pairM.datum.getx() << "y= " << current->pairM.datum.gety() << "label = " << current->pairM.datum.get\_label() << endl;

current = current->nextM;

}

std::cout << std::endl;

}

bool LookupTable::isEmpty() const

{

return sizeM == 0;

}

int \*LookupTable::retrieve\_at(int i)

{

if (i < 0 || i >= sizeM)

{

return nullptr;

}

LT\_Node \*current = headM;

int count = 0;

while (current != nullptr && count < i)

{

current = current->nextM;

count++;

}

if (current != nullptr)

{

return &(current->pairM.key);

}

else

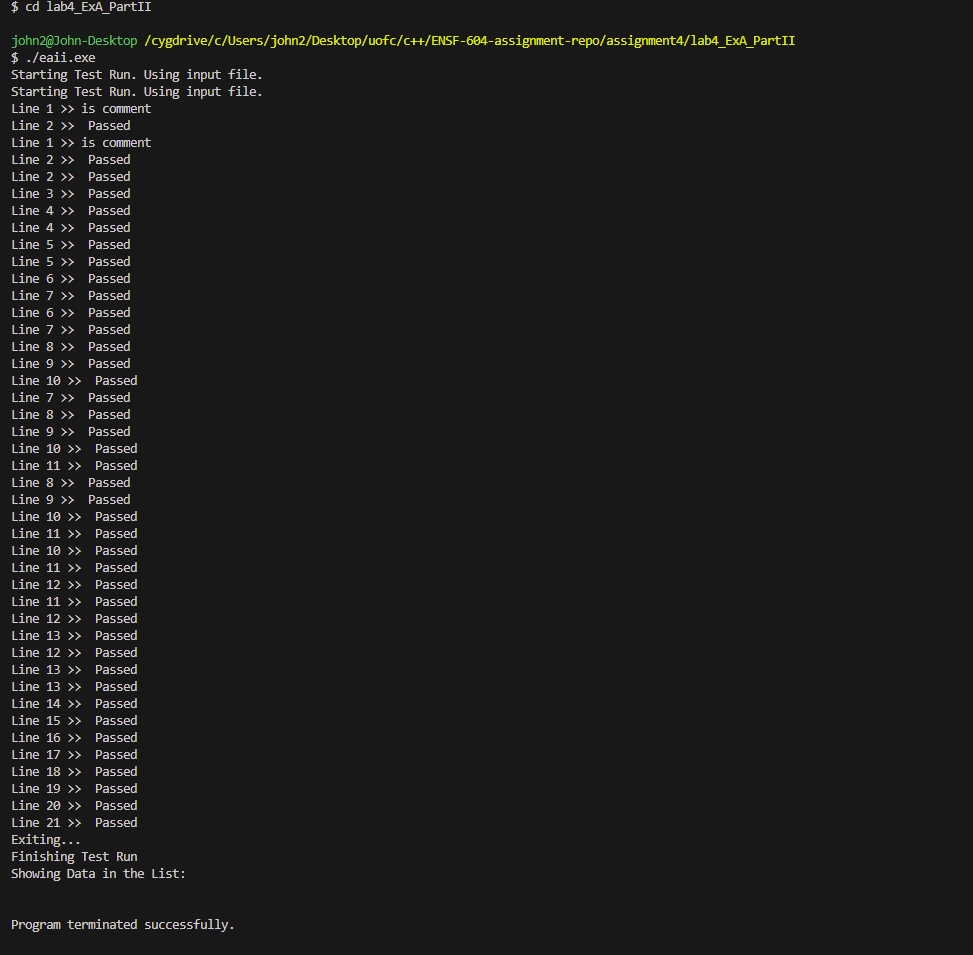
{

return nullptr;

}

}

Execution output:



# Exercise B

void write\_binary\_file(City cities[], int size, char \*filename)

{

    ofstream stream(filename, ios::out | ios::binary);

    if (stream.fail())

    {

        cerr << "Failed to open file" << filename << endl;

        exit(1);

    }

    for (int i = 0; i < size; i++)

    {

        char\* bytePtr = (char\*)&cities[i];

        stream.write(bytePtr, sizeof(City));

    }

    stream.close();

}

void print\_from\_binary(char \*filename)

{

    ifstream stream(filename, ios::in | ios::binary);

    if (stream.fail())

    {

        cerr << "Failed to open file" << filename << endl;

        exit(1);

    }

    City city;

    char\* bytePtr = (char\*)&city;

    while (stream.read(bytePtr, sizeof(City)))

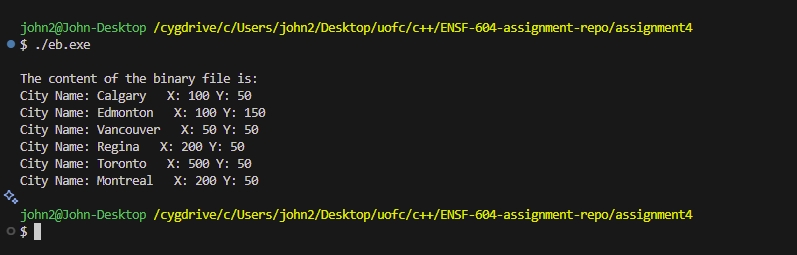
    {

        cout << "City Name: " << city.name << "   X: " << city.x << " Y: " << city.y << endl;

    }

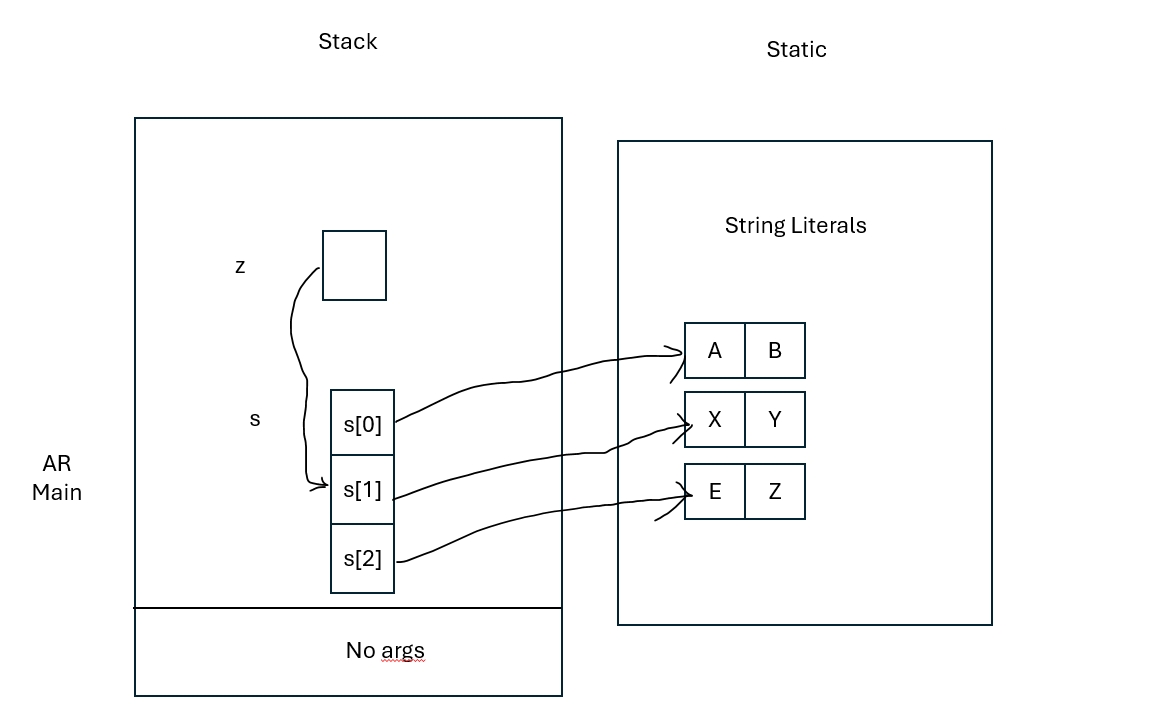
    stream.close();

}

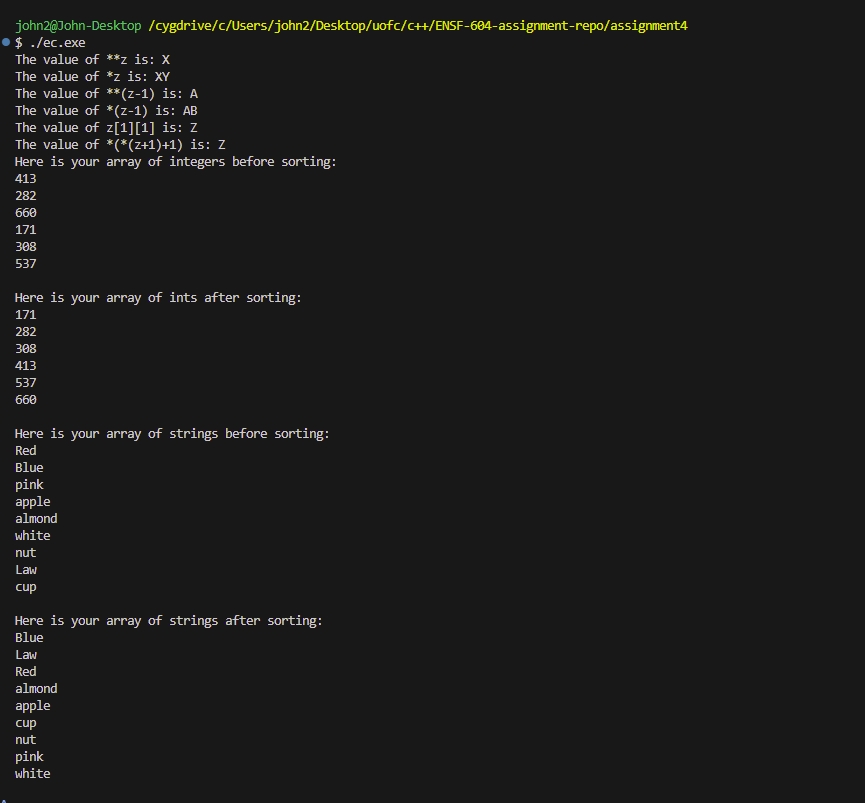
Execution output

# Exercise C

AR diagram



Program output



Code

/\*

 \* lab4exe\_C.cpp

 \* ENSF 694 - Lab 4 Exercise C

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#include <cstring>

#include <iostream>

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;

    // 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

    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)

{

    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\* pointer\_to\_insert;

    for (i = 1; i < n; i++) {

        pointer\_to\_insert = str\_array[i];

        /\* Shift values greater than pointer\_to\_insert. \*/

        j = i;

        while ( j > 0 && strcmp(str\_array[j - 1], pointer\_to\_insert) > 0) {

            str\_array[j] = str\_array[j - 1];

            j--;

        }

        str\_array[j] = pointer\_to\_insert;

    }

}

# Exercise D

Code

/\*

 \*  matrix.cpp

 \* ENSF 694 - Lab 4 Exercise D

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#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

{

    // COMMENT OUT THE FOLLOWING LINE AND COMPLETE THE DEFINITION OF THIS FUNCTION

    // cout << "\nSorry I don't know how to calculate sum of rowsM in a matrix. ";

        for (int i = 0; i < rowsM; i++)

    {

        double sum = 0.0;

        for (int j = 0; j < colsM; j++)

        {

            sum += matrixM[i][j];

        }

        sum\_rowsM[i] = sum;

    }

}

void Matrix::sum\_of\_cols() const

{

    // COMMENT OUT THE FOLLOWING LINE AND COMPLETE THE DEFINITION OF THIS FUNCTION

    // cout << "\nSorry I don't know how to calculate sum of columns in a matrix. ";

        for (int j = 0; j < colsM; j++)

    {

        double sum = 0.0;

        for (int i = 0; i < rowsM; i++)

        {

            sum += matrixM[i][j];

        }

        sum\_colsM[j] = sum;

    }

}

void Matrix::copy(const Matrix &source)

{

    // THIS FUNCITON IS DEFECTIVE AND DOSEN'T PROPERLY MAKE THE COPY OF SROUCE

    if (source.matrixM == NULL)

    {

        matrixM = NULL;

        sum\_rowsM = NULL;

        sum\_colsM = NULL;

        rowsM = 0;

        colsM = 0;

        return;

    }

    rowsM = source.rowsM;

    colsM = source.colsM;

    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 i = 0; i < rowsM; i++)

    {

        for (int j = 0; j < colsM; j++)

        {

            matrixM[i][j] = source.matrixM[i][j];

        }

    }

    sum\_rowsM = new double[rowsM];

    assert(sum\_rowsM != NULL);

    sum\_colsM = new double[colsM];

    assert(sum\_colsM != NULL);

    for (int i = 0; i < rowsM; i++)

    {

        sum\_rowsM[i] = source.sum\_rowsM[i];

    }

    for (int j = 0; j < colsM; j++)

    {

        sum\_colsM[j] = source.sum\_colsM[j];

    }

    // STUDENTS MUST COMMENT OUT THE FOLLOWING LINE AND FIX THE FUNCTION'S PROBLEM

    // cout << "\nSorry copy fucntion is defective. ";

}

void Matrix::destroy()

{

    // COMMENT OUT THE FOLLOWING LINE AND COMPLETE THE DEFINITION OF THIS FUNCTION

    for (int i = 0; i < rowsM; i++)

    {

        delete[] matrixM[i];

    }

    delete[] matrixM;

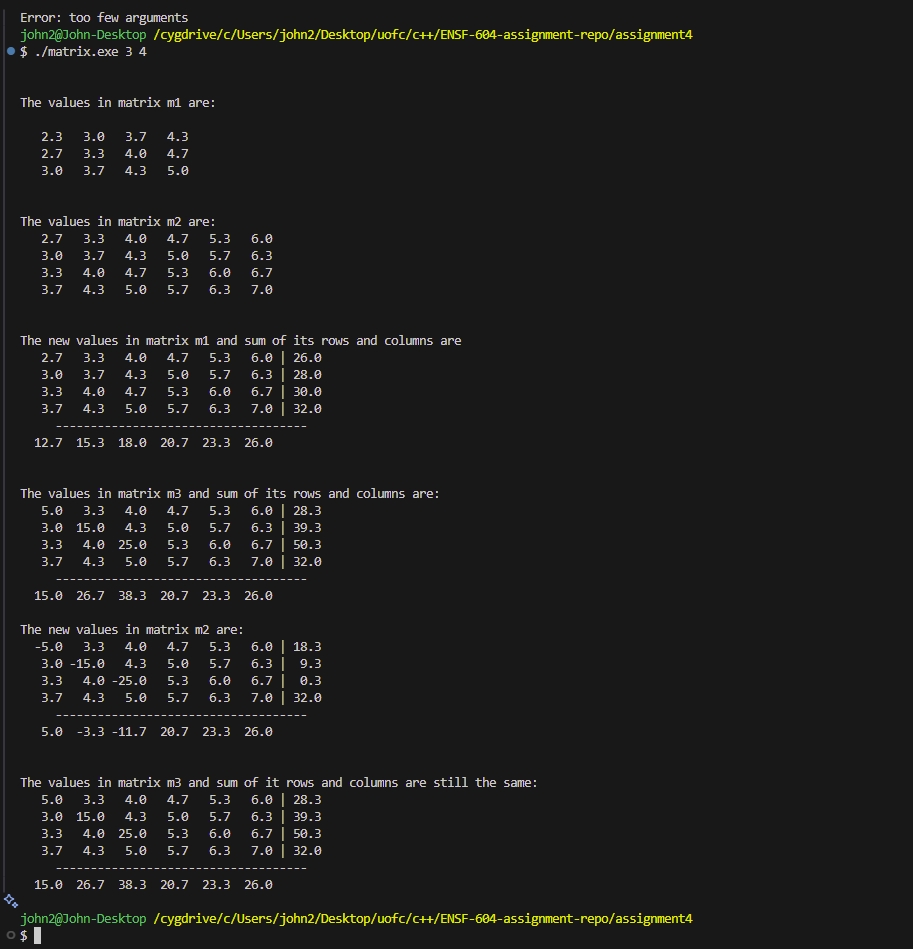
    delete[] sum\_rowsM;

    delete[] sum\_colsM;

    // cout << "\nProgram ended without destroying matrices.\n";

}

Execution output



# Exercise E a

Hashtable.cpp

/\*

 \* HashTable.cpp

 \* ENSF 694 - Lab 4 Exercise E

 \*  Created by Mahmood Moussavi

 \*  Completed by: John Zhou

 \*/

#include "HashTable.h"

#include "Node.h"

#include "List.h"

#include "Flight.h"

#include <iostream>

#include <fstream>

#include <string>

using namespace std;

HashTable::HashTable(int size)

{

    tableSize = size;

    totalRecords = 0;

    nonCollisionCount = 0;

    table = new List \*[tableSize];

    for (int i = 0; i < tableSize; ++i)

    {

        table[i] = nullptr;

    }

}

HashTable::~HashTable()

{

    for (int i = 0; i < tableSize; ++i)

    {

        delete table[i];

    }

    delete[] table;

}

int HashTable::hashFunction(const string &flightID) const

{

    int hashValue = 0;

    for (char ch : flightID)

    {

        hashValue += (int)ch;

    }

    return hashValue % tableSize;

}

void HashTable::insert(const Flight &flight)

{

    int index = hashFunction(flight.getFlightID());

    if (table[index] == nullptr)

    {

        table[index] = new List();

    }

    table[index]->insert(flight);

    totalRecords++;

    if (table[index]->getSize() == 1)

    {

        nonCollisionCount++;

    }

}

Flight \*HashTable::search(const string &flightID) const

{

    int index = hashFunction(flightID);

    if (table[index] != nullptr)

    {

        Node \*foundNode = table[index]->search(flightID);

        if (foundNode != nullptr)

        {

            return &foundNode->data;

        }

    }

    return nullptr;

}

void HashTable::printTable() const

{

    // Printing hash table statistics

    cout << "Total records: " << getTotalRecords() << endl;

    cout << "Table size: " << this->getTableSize() << endl;

    cout << "Packing density: " << getPackingDensity() << endl;

    cout << "Table density: " << this->getTableDensity() << endl;

    cout << "Hash efficiency: " << this->getHashEfficiency() << endl;

    for (int i = 0; i < tableSize; ++i)

    {

        if (table[i] == nullptr)

        {

            cout << "bucket " << i << ": is empty \n";

        }

        else

        {

            cout << "bucket " << i << ": ";

            table[i]->printList();

        }

    }

}

double HashTable::getNonCollisionEfficiency() const

{

    return (double)nonCollisionCount / totalRecords;

}

int HashTable::calculateTotalSearchCost() const

{

    int totalCost = 0;

    for (int i = 0; i < tableSize; ++i)

    {

        if (table[i] != nullptr)

        {

            totalCost += table[i]->getSize();

        }

    }

    return totalCost;

}

double HashTable::getTableDensity() const

{

    int nonEmptyBuckets = 0;

    for (int i = 0; i < tableSize; ++i)

    {

        if (table[i] != nullptr)

        {

            nonEmptyBuckets++;

        }

    }

    return (double)(nonEmptyBuckets) / tableSize;

}

double HashTable::getPackingDensity() const

{

    return (double)totalRecords / tableSize;

}

double HashTable::getHashEfficiency() const

{

    double packingDensity = getPackingDensity();

    double avgReads = (double)calculateTotalSearchCost() / totalRecords;

    return packingDensity / avgReads;

}

read\_flight\_info

void read\_flight\_info(ifstream &fin, vector<Flight> &flights) {

    string line;

    while (getline(fin, line)) {

        if (line.empty())

            continue;

        istringstream iss(line);

        string flightID, origin, destination, depDate, depTime;

        int capacity;

        // You can adjust the reading logic if fields are fixed width.

        if (!(iss >> flightID >> origin >> destination >> depDate >> depTime >> capacity)) {

            cerr << "Malformed record: " << line << endl;

            continue;

        }

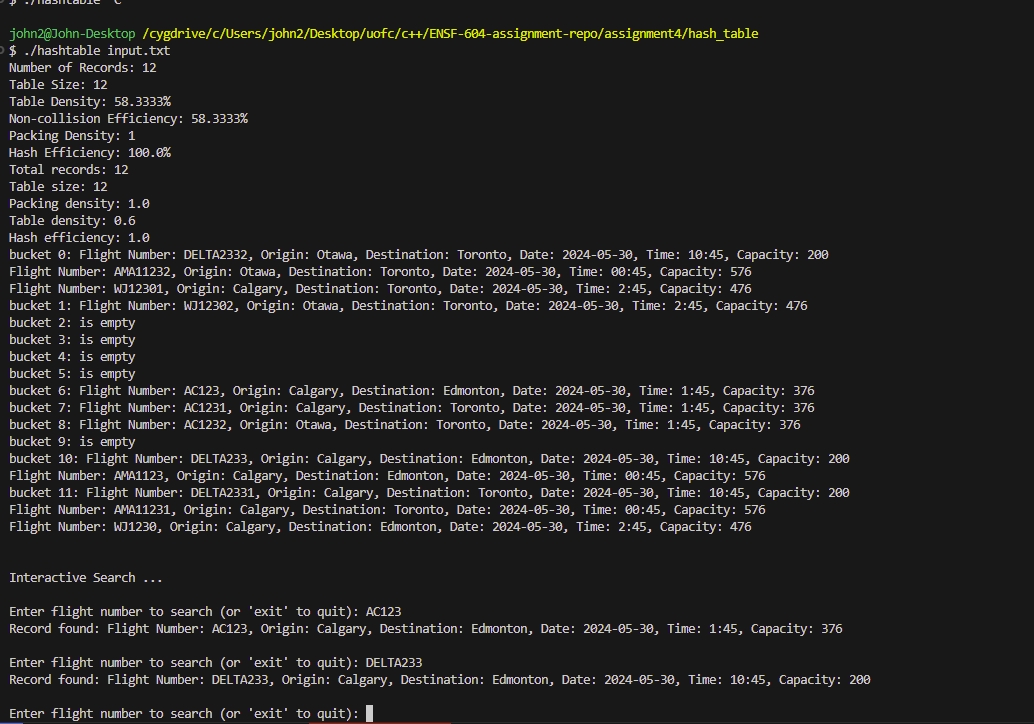
        Flight flight(flightID, origin, destination, depDate, depTime, capacity);

        flights.push\_back(flight);

    }

}

# Exercise E b



# Exercise E c

double HashTable::getNonCollisionEfficiency() const

{

    return (double)nonCollisionCount / totalRecords;

}

int HashTable::calculateTotalSearchCost() const

{

    int totalCost = 0;

    for (int i = 0; i < tableSize; ++i)

    {

        if (table[i] != nullptr)

        {

            totalCost += table[i]->getSize();

        }

    }

    return totalCost;

}

double HashTable::getTableDensity() const

{

    int nonEmptyBuckets = 0;

    for (int i = 0; i < tableSize; ++i)

    {

        if (table[i] != nullptr)

        {

            nonEmptyBuckets++;

        }

    }

    return (double)(nonEmptyBuckets) / tableSize;

}

double HashTable::getPackingDensity() const

{

    return (double)totalRecords / tableSize;

}

double HashTable::getHashEfficiency() const

{

    double packingDensity = getPackingDensity();

    double avgReads = (double)calculateTotalSearchCost() / totalRecords;

    return packingDensity / avgReads;

}

# Exercise E d

hashIndex = (sum of ASCII codes of characters in flightID) % tableSize

this has poor distribution. ABC123 and 123ABC will result in the same spot.

Using Carter-Wegman Hashing can introduce randomness into the hashing function which will have a better distribution.