Course: ENSF 694 – Summer 2025
Lab #: 04
Instructor: Dr. Mahmood Moussavi
Student Name: Jack Shenfield
Submission Date: Wednesday, July 30th, 2025

Exercise A Part I

Source Code (lookupTable.cpp):

```
#include "lookupTable.h"
#include <iostream>


/*
 *  lookupTable.cpp
 *  ENSF 694 Lab 4, exercise A
 *  Created by Mahmood Moussavi
 *  Completed by: Jack Shenfield
 *  Development Date: July 29th, 2025
 */

// constructor for LT_Node struct
LT_Node::LT_Node(const Pair& pairA, LT_Node *nextA):pairM(pairA), nextM(nextA){}

// initialize all member variables to 0
LookupTable::LookupTable(){
    sizeM = 0;
    headM = nullptr;
    cursorM = nullptr;
}

// copy ctor
LookupTable::LookupTable(const LookupTable  & source):sizeM(source.sizeM), headM(nullptr), cursorM(nullptr){
    if(source.headM == nullptr){
        return; // list to be copied is empty
    }

    // copy head node into headM
    headM = new LT_Node(source.headM->pairM, nullptr);
    LT_Node * src = source.headM->nextM; // track old list in iteration
    LT_Node * dest = headM; // track new list

    // point cursor to head
```

```
    cursorM = headM;

    while(src != nullptr){
        dest->nextM = new LT_Node(src->pairM, nullptr); // create new node using src data
        dest = dest->nextM; // update pointer to new node
        if(src == source.cursorM){
            cursorM = dest;
        }
        src = src->nextM;
    }
}



// assignment operator
LookupTable& LookupTable::operator =(const LookupTable& rhs){
    // if it is the same object return it
    if (this == &rhs){
        return *this;
    }

    // reset to initial values
    sizeM = rhs.sizeM;
    headM = nullptr;
    cursorM = nullptr;

    // If it is empty, return it
    if (rhs.headM == nullptr){
        return *this;
    }

    // Create new node
    headM = new LT_Node(rhs.headM->pairM, nullptr);

    LT_Node* src = rhs.headM->nextM;
    LT_Node* dest = headM;
```

```
    cursorM = headM;


    // Loop through and create new list
    while (src != nullptr) {
        dest->nextM = new LT_Node(src->pairM, nullptr);
        dest = dest->nextM;
        if (src == rhs.cursorM) {
            cursorM = dest;
        }
        src = src->nextM;
    }

    return *this;
}

// dtor
LookupTable::~LookupTable(){
    make_empty(); // call the make empty function created later on.
}
LookupTable& LookupTable::begin(){
    cursorM = headM;
    return *this;
}

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

int LookupTable::cursor_ok() const{
    if(cursorM == nullptr){ // pointing at nothing, return 0
        return 0;
    }
    else{ // if not pointing at nothing, assume pointing at list
        return 1;
    }
}
```

```
const int& LookupTable::cursor_key() const{
    if(cursor_ok() == 1){
        return cursorM->pairM.key;
    }
}

const Type& LookupTable::cursor_datum() const{
    if(cursor_ok() == 1){
        return cursorM->pairM.datum;
    }
}

// inserts keys in order
void LookupTable::insert(const Pair& pairA) {
    cursorM = headM;
    LT_Node* prev = nullptr;

    // Check if key already exists — if so, update the datum
    while (cursorM != nullptr) {
        if (cursorM->pairM.key == pairA.key) {
            cursorM->pairM.datum = pairA.datum;
            cursorM = nullptr;
            return;
        }
        if (cursorM->pairM.key > pairA.key) {
            break;  // found insert position
        }
        prev = cursorM;
        cursorM = cursorM->nextM;
    }

    // Create new node to insert
    LT_Node* newNode = new LT_Node(pairA, cursorM);

    if (prev == nullptr) {
        // Insert at head
```

```
        headM = newNode;
    } else {
        prev->nextM = newNode;
    }

    cursorM = nullptr;
    ++sizeM;
}




int LookupTable::remove(const int& keyA){

    cursorM = nullptr; // off-list

    LT_Node* prev = nullptr;
    cursorM = headM;

    while (cursorM != nullptr) {
        if (cursorM->pairM.key == keyA) {
            // Remove node
            if (prev == nullptr) {
                // Node head
                headM = cursorM->nextM;
            } else {
                // Node is not at head
                prev->nextM = cursorM->nextM;
            }

            int removed = cursorM->pairM.key;
            delete cursorM;
            cursorM = nullptr;
            --sizeM;
            return removed; // return the removed key
        }
```

```
        prev = cursorM;
        cursorM = cursorM->nextM;
    }

    cursorM = nullptr;

    return 0; // key not found
}

void LookupTable::find(const int& keyA){

    cursorM = headM;

    // search for key
    while(cursorM != nullptr){
        if(cursorM->pairM.key == keyA){
            return;
        }
        cursorM = cursorM->nextM;
    }

    cursorM = nullptr;
}

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

void LookupTable::step_fwd(){
    if(cursorM->nextM == nullptr || cursor_ok() == 0){
        cursorM = nullptr;
    }
```

```
    else{
        cursorM = cursorM->nextM;
    }
}

void LookupTable::make_empty(){

    // delete each node from memory going down the list
    LT_Node* current = headM;
    while (current != nullptr) {
        LT_Node* temp = current;
        current = current->nextM;
        delete temp;
    }

    // reset variables
    headM = nullptr;
    cursorM = nullptr;
    sizeM = 0;
}


void LookupTable::display()const{
    LT_Node* ptr = headM;

    // print lines
    while (ptr != nullptr) {
        std::cout << "[" << ptr->pairM.key << ": " << ptr->pairM.datum << "] ";
        ptr = ptr->nextM;
    }
    std::cout << std::endl; // endline
}


bool LookupTable::isEmpty()const{
    if(sizeM == 0){
        return true;
```

```
    }
    else{
        return false;
    }
}

int LookupTable::retrieve_at(int i){
    if(i < 0 || i >= sizeM){
        throw std::out_of_range("Index DNE"); // index doesn't exist
    }

    cursorM = headM;
    int index = 0;

    while(cursorM != nullptr && index < i){
        cursorM = cursorM->nextM;
        index++;
    }

    if(cursorM != nullptr){
        return cursorM->pairM.key;
    }

}
```

Output:

```
(base) jbs@Jacks-MacBook-Air Lab04_ExA_Part1 % g++ -std=c++17
lookupTable.cpp lookupTable_tester-part1.cpp -o tester
lookupTable.cpp:114:1: warning: non-void function does not return a value in all
control paths [-Wreturn-type]
  114 | }
      | ^
lookupTable.cpp:120:1: warning: non-void function does not return a value in all
control paths [-Wreturn-type]
```

```
  120 | }
      | ^
lookupTable.cpp:280:1: warning: non-void function does not return a value in all
control paths [-Wreturn-type]
  280 | }
      | ^
3 warnings generated.
(base) jbs@Jacks-MacBook-Air Lab04_ExA_Part1 % ./tester
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]
[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] [8002: Joe Morrison]

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

Test copying: keys should be 8001, and 8002
[8001: Tim Hardy] [8002: Joe Morrison]
[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 Lookup Table <not template>-----***
PRESS RETURN TO CONTINUE.

Program terminated successfully.

(base) jbs@Jacks-MacBook-Air Lab04_ExA_Part1 %
```

Exercise A Part II

Source Code:

Source code for the changed files is attached.

Output:

```
(base) jbs@Jacks-MacBook-Air Lab04_ExA_Part2 % g++ -std=c++17 Point.cpp
lookupTable.cpp lookupTable_tester_part2.cpp -o tester

lookupTable.cpp:115:1: warning: non-void function does not return a value in all
control paths [-Wreturn-type]
  115 | }
      | ^
lookupTable.cpp:121:1: warning: non-void function does not return a value in all
control paths [-Wreturn-type]
  121 | }
      | ^
lookupTable.cpp:281:1: warning: non-void function does not return a value in all
control paths [-Wreturn-type]
  281 | }
      | ^
3 warnings generated.
(base) jbs@Jacks-MacBook-Air Lab04_ExA_Part2 % ./tester
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.

(base) jbs@Jacks-MacBook-Air Lab04_ExA_Part2 %
```

Exercise B

Fcn Definition:

```
void print_from_binary(char* filename){


    ifstream stream(filename, ios::in | ios::binary); // same as above, but for input file stream
    if (stream.fail()) {
        cerr << "failed to open file: " << filename << endl;
        exit(1);
    }



    City city;

    while (stream.read(reinterpret_cast<char*>(&city), sizeof(City))) { // print all
        cout << "City: " << city.name << ", x: " << city.x << ", y: " << city.y << endl;
    }

    stream.close();
}
```

Output:

```
se) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % ./lab4
```

```
The content of the binary file is:
City: Calgary, x: 100, y: 50
City: Edmonton, x: 100, y: 150
City: Vancouver, x: 50, y: 50
City: Regina, x: 200, y: 50
City: Toronto, x: 500, y: 50
City: Montreal, x: 200, y: 50
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 %
```

Exercise C

Source Code:

```
/*
 *  lab4exe_C.cpp
 *  ENSF 694 Lab 4, exercise C
 *  Created by Mahmood Moussavi
 *  Completed by: Jack Shenfield
 *  Development Date: July 30th, 2025
 */

#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
```

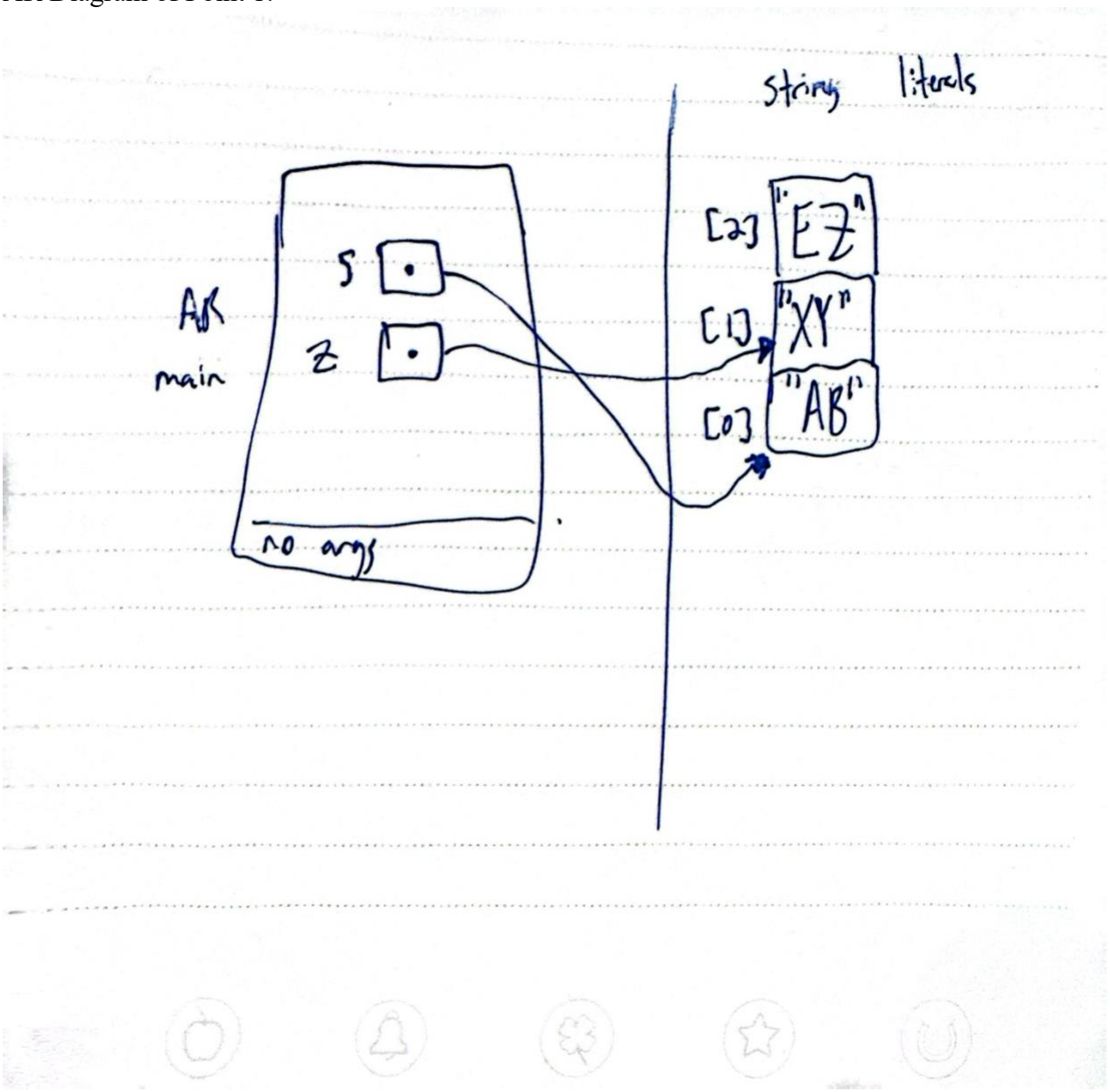
```
*   n > 0.
*   Array elements str_array[0] ... str_array[n - 1] exist.
* PROMISES
*   pointers in str_array are rearranged so that strings:
*   str_array[0] points to a string with the smallest string (lexicographicall) ,
*   str_array[1] points to the second smallest string, ..., str_array[n-2]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



    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){ // same logic as above, but changed to strings.

    int i;
    int j;
    const char* str_to_insert;

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

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

        str_array[j] = str_to_insert;
    }
}

/* 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
*/
```

AR Diagram of Point 1:



Output:

```
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % ./lab4_C
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:
Red
nut
Law
cup
Blue
pink

```
apple
white
almond

(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 %
```

Exercise D

Source Code:

```
//   matrix.cpp


#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
{

    for(int i = 0; i < rowsM; i++){
        sum_rowsM[i] = 0; // initialize sum at 0
        for(int j = 0; j < colsM; j++)
            sum_rowsM[i] += matrixM[i][j]; // sum each data point in the row
    }
    // 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. ";
```

```
}

void Matrix::sum_of_cols()const
{
    // same logic as above, but swap rows/cols
    for(int i = 0; i < colsM; i++){
        sum_colsM[i] = 0; // initialize sum at 0
        for(int j = 0; j < rowsM; j++)
            sum_colsM[i] += matrixM[j][i]; // sum each data point in the row
    }
    // 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. ";

}

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;

    sum_rowsM = new double[rowsM];
    assert(sum_rowsM != NULL);

    for (int i = 0; i < rowsM; ++i) { // initialize proper row sums
        sum_rowsM[i] = source.sum_rowsM[i];
    }
```

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

    for (int j = 0; j < colsM; ++j) { // initialize proper column sums
        sum_colsM[j] = source.sum_colsM[j];
    }

    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];
        }
    }
    // STUDENTS MUST COMMENT OUT THE FOLLOWING LINE AND FIX THE FUNCTION'S PROBLEM
    //cout << "\nSorry copy fucntion is defective. ";
}



void Matrix::destroy()
{
    // destroy each row
    for(int i = 0; i < rowsM; i++){
        delete [] matrixM[i];
    }

    // delete leftover array
    delete [] matrixM;

    // reset member variables
    matrixM = nullptr;
    rowsM = 0;
```

```
    colsM = 0;

    // COMMENT OUT THE FOLLOWING LINE AND COMPLETE THE DEFINITION OF THIS FUNCTION
    //cout << "\nProgram ended without destroying matrices.\n";
}
```

Output:

```
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % g++ matrix.cpp
lab4exe_D.cpp -o matrix
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % ./matrix

Error: too few arguments%                                    (base)
jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % ./matrix 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
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 %
```

Exercise E

a)

```
/*
 * lab4exe_E.cpp
 * ENSF 694 Lab 4, exercise E
 * Created by Mahmood Moussavi
 * Completed by: Jack Shenfield
 * Development Date: July 30th, 2025
```

```
 */


#include "HashTable.h"

// function definitions


int HashTable::hashFunction(const std::string &flightID) const{ // inspired by chatGPT
    unsigned int hash = 0;

    for (char c : flightID) {
        hash = hash * 31 + c;  // prime number multiplier is 31
    }
    return hash % tableSize; //
}

// constructor, initialize most values at 0. size at inputted size.
HashTable::HashTable(int size):tableSize(size), totalRecords(0), nonCollisionCount(0), elementsUsed(0) {

    table = new List*[tableSize];
    for (int i = 0; i < tableSize; ++i)
        table[i] = nullptr; // fill table with null pointers
}

HashTable::~HashTable(){
    for (int i = 0; i < tableSize; ++i) { // delete each value

        if (table[i] != nullptr)
            delete table[i];
    }

    delete[] table; // delete the leftover empty table

}

void HashTable::insert(const Flight &flight){
```

```
    int ind = hashFunction(flight.getFlightID()); // compute hash # (useable index) with my prime function

    if (table[ind] == nullptr) { // If there is no value at this index

        table[ind] = new List(); // create a new List at the computed hash #
        table[ind]->insert(flight); // insert the flight into that list

        nonCollisionCount++; // There was no collision here. increment
        elementsUsed++; // new element

    } else {
        table[ind]->insert(flight); // there is already a flight here, chain to previous one.
    }

    totalRecords++; // increment total records

}

Flight* HashTable::search(const std::string &flightID)const{

    int index = hashFunction(flightID); // compute hash # from inputted flight ID

    if (table[index] == nullptr){
        return nullptr; // If there is nothing at the index, return nullptr
    }
    Node* result = table[index]->search(flightID);
    return result ? &result->data : nullptr; // return the result or nullptr depending on result
}

void HashTable::printTable() const{

    for (int i = 0; i < tableSize; ++i) {
        std::cout << "Chain " << i << ": ";
        if (table[i]) // print if there is values
            table[i]->printList();
        else // else, print empty
```

```
            std::cout << "Empty";
        std::cout << std::endl;
    }
}

double HashTable::getNonCollisionEfficiency() const{

    if (totalRecords == 0){
        return 0.0; // no values, return nothing.
    }
    // otherwise, return non collision %
    return ((double)nonCollisionCount / totalRecords * 100.0);
}


int HashTable::calculateTotalSearchCost()const{
    int totalCost = 0; // initialize at 0;

    for (int i = 0; i < tableSize; ++i) {
        List* chain = table[i]; //
        if (chain != nullptr) {

            Node* current = chain->getHead(); // point at head index of chain

            int position = 1;

            while (current != nullptr) { // for each real value
                totalCost += position;  // search cost is the position
                current = current->next; // move to next value
                ++position; // increment

            }
        }

    }
```

```
    return totalCost;
}

double HashTable::getTableDensity() const{
    return(static_cast<double>(elementsUsed) / tableSize); // number of elements used / total table size
}

double HashTable::getPackingDensity() const{
    return(static_cast<double>(totalRecords) / tableSize); // number of total records / table size
}

double HashTable::getHashEfficiency() const{

    if (totalRecords == 0) return 0.0; // if empty, return 0.0

    return(static_cast<double>(calculateTotalSearchCost()) / totalRecords); // otherwise, return search cost / total table
size
}
```

b)

```
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % g++ -std=c++17
HashTable.cpp Flight.cpp List.cpp HashTable_tester.cpp -o hashtest
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 % ./hashtest input.txt
Number of Records: 12
Table Size: 12
Table Density: 75%
Non-collision Efficiency: 7500%
Packing Density: 1
Hash Efficiency: 133.3%
Chain 0: Flight Number: AMA11232, Origin: Otawa, Destination: Toronto, Date:
2024-05-30, Time: 00:45, Capacity: 576
Flight Number: WJ12301, Origin: Calgary, Destination: Toronto, Date: 2024-05-30,
Time: 2:45, Capacity: 476
Flight Number: AC123, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30,
Time: 1:45, Capacity: 376
```

Chain 1: Flight Number: WJ12302, Origin: Otawa, Destination: Toronto, Date: 2024-05-30, Time: 2:45, Capacity: 476
Flight Number: AC1231, Origin: Calgary, Destination: Toronto, Date: 2024-05-30, Time: 1:45, Capacity: 376

Chain 2: Flight Number: AC1232, Origin: Otawa, Destination: Toronto, Date: 2024-05-30, Time: 1:45, Capacity: 376

Chain 3: Flight Number: DELTA2331, Origin: Calgary, Destination: Toronto, Date: 2024-05-30, Time: 10:45, Capacity: 200

Chain 4: Flight Number: DELTA2332, Origin: Otawa, Destination: Toronto, Date: 2024-05-30, Time: 10:45, Capacity: 200

Chain 5: Flight Number: WJ1230, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 2:45, Capacity: 476

Chain 6: Flight Number: AMA1123, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 00:45, Capacity: 576

Chain 7: Empty
Chain 8: Empty
Chain 9: Empty
Chain 10: Flight Number: DELTA233, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 10:45, Capacity: 200

Chain 11: Flight Number: AMA11231, Origin: Calgary, Destination: Toronto, Date: 2024-05-30, Time: 00:45, Capacity: 576

Interactive Search ...

Enter flight number to search (or 'exit' to quit): exit
(base) jbs@Jacks-MacBook-Air ENSF694_LabAssignment4 %

c)

```
double HashTable::getPackingDensity() const{
    return(static_cast<double>(totalRecords) / tableSize); // number of total records / table size
}

double HashTable::getHashEfficiency() const{

    if (totalRecords == 0) return 0.0; // if empty, return 0.0

    return(static_cast<double>(calculateTotalSearchCost()) / totalRecords); // otherwise, return search cost / total table
size
}
```

d)

The technique I chose multiplies the current hash by a prime number (31) and then adds the current character in the string (ASCII value) to that number, iteratively. I found it by looking up "simple hash functions for c++", consulting AI, and reading the lecture slides. It could be improved by potentially using a more complicated hash function that spread the data more evenly, or resulted in less collisions.