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

```
$ ./ea.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

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.
```

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:

```
$ cd lab4_ExA_PartII

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment4/lab4_ExA_PartII
$ ./eaii.exe
Starting Test Run. Using input file.
Starting Test Run. Using input file.
Line 1 >> is comment
Line 2 >>  Passed
Line 1 >> is comment
Line 2 >>  Passed
Line 2 >>  Passed
Line 3 >>  Passed
Line 4 >>  Passed
Line 4 >>  Passed
Line 5 >>  Passed
Line 5 >>  Passed
Line 6 >>  Passed
Line 7 >>  Passed
Line 6 >>  Passed
Line 7 >>  Passed
Line 8 >>  Passed
Line 9 >>  Passed
Line 10 >>  Passed
Line 7 >>  Passed
Line 8 >>  Passed
Line 9 >>  Passed
Line 10 >>  Passed
Line 11 >>  Passed
Line 8 >>  Passed
Line 9 >>  Passed
Line 10 >>  Passed
Line 11 >>  Passed
Line 10 >>  Passed
Line 11 >>  Passed
Line 12 >>  Passed
Line 11 >>  Passed
Line 12 >>  Passed
Line 13 >>  Passed
Line 12 >>  Passed
Line 13 >>  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.
```

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);
   }
```

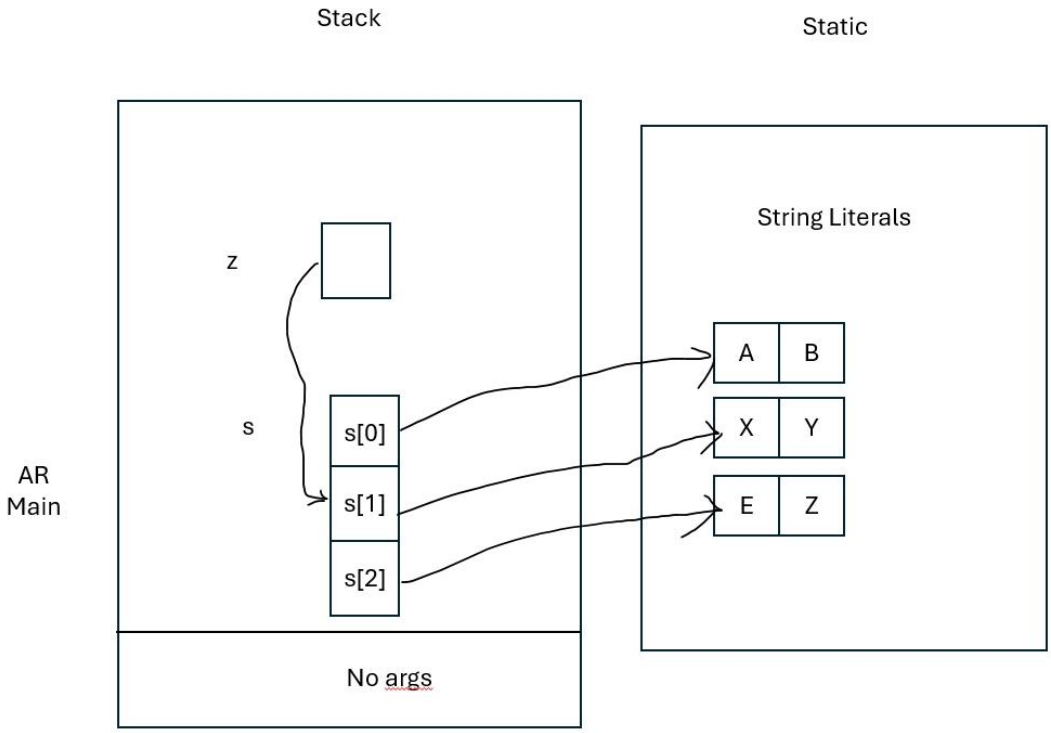
```
    City city;
    char* bytePtr = (char*)&city;
    while (stream.read(bytePtr, sizeof(City)))
    {
        cout << "City Name: " << city.name << "   X: " << city.x << " Y: " << city.y << endl;
    }

    stream.close();
}
```

Execution output

Exercise C

AR diagram



Program output

```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment4
$ ./ec.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
```

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

```
Error: too few arguments
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment4
$ ./matrix.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

john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment4
$
```

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

```
john2@John-Desktop /cygdrive/c/Users/john2/Desktop/uofc/c++/ENSF-604-assignment-repo/assignment4/hash_table
$ ./hashtable input.txt
Number of Records: 12
Table Size: 12
Table Density: 58.3333%
Non-collision Efficiency: 58.3333%
Packing Density: 1
Hash Efficiency: 100.0%
Total records: 12
Table size: 12
Packing density: 1.0
Table density: 0.6
Hash efficiency: 1.0
bucket 0: Flight Number: DELTA2332, Origin: Otawa, Destination: Toronto, Date: 2024-05-30, Time: 10:45, Capacity: 200
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
bucket 1: Flight Number: WJ12302, Origin: Otawa, Destination: Toronto, Date: 2024-05-30, Time: 2:45, Capacity: 476
bucket 2: is empty
bucket 3: is empty
bucket 4: is empty
bucket 5: is empty
bucket 6: Flight Number: AC123, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 1:45, Capacity: 376
bucket 7: Flight Number: AC1231, Origin: Calgary, Destination: Toronto, Date: 2024-05-30, Time: 1:45, Capacity: 376
bucket 8: Flight Number: AC1232, Origin: Otawa, Destination: Toronto, Date: 2024-05-30, Time: 1:45, Capacity: 376
bucket 9: is empty
bucket 10: Flight Number: DELTA233, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 10:45, Capacity: 200
Flight Number: AMA1123, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 00:45, Capacity: 576
bucket 11: Flight Number: DELTA2331, Origin: Calgary, Destination: Toronto, Date: 2024-05-30, Time: 10:45, Capacity: 200
Flight Number: AMA11231, Origin: Calgary, Destination: Toronto, Date: 2024-05-30, Time: 00:45, Capacity: 576
Flight Number: WJ1230, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 2:45, Capacity: 476


Interactive Search ...

Enter flight number to search (or 'exit' to quit): AC123
Record found: Flight Number: AC123, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 1:45, Capacity: 376

Enter flight number to search (or 'exit' to quit): DELTA233
Record found: Flight Number: DELTA233, Origin: Calgary, Destination: Edmonton, Date: 2024-05-30, Time: 10:45, Capacity: 200

Enter flight number to search (or 'exit' to quit):
```

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.