Course: Programming Fundamental – ENSF 337
Lab #: Lab 9
Instructor: M. Moussavi
Student Name: Aarushi Roy Choudhury
Lab Section: B01
Date submitted: Dec, 3 2021

Exercise B

```
void print_from_binary(char* filename) {
    /* Students must complete the implementation of this file. */

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

    if (stream.fail())
    {
        cerr << "failed to open file: " << filename << endl;
        exit(1);
    }

    City city[6];

    for(int i = 0; i < 6; i++)
        stream.read((char*) &city[i], sizeof(City));

    stream.close();


    if(!stream.good())
    {
        cout<<"Error occured at reading time!"<<endl;
        exit(1);
    }

    for(int i = 0; i < 6; i++) {
        cout<<"Name: "<<city[i].name<<", x coordinate: "<<city[i].x<<", y
coordinate: "<<city[i].y<<endl;
    }
}
```

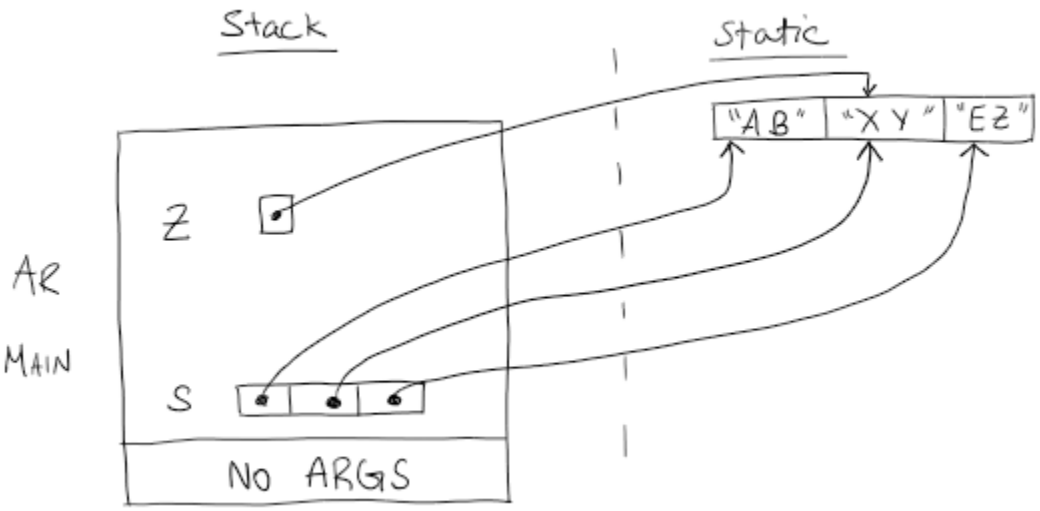
```
C:\Users\Aarus\Desktop\Lab9>g++ -std=c++11 -Wall lab9ExB.cpp

C:\Users\Aarus\Desktop\Lab9>a.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

C:\Users\Aarus\Desktop\Lab9>
```

Exercise C

```
String_Vector transpose (const String_Vector& sv) {

// STUDENTS MUST COMPLETE THE DEFINITION OF THIS FUNCTION.

const int ROWS = sv.size();

if(ROWS == 0)
    return sv;

const int COLS = sv.at(0).size();

String_Vector vs(COLS);

for(int i = 0; i < ROWS; i++) {
    for(int j = 0; j < COLS; j++) {
        vs[j].push_back(sv[i][j]);
    }
}
cout<<"Transposed Vector:"<<endl;
return vs;

}
```

```
PS C:\Users\Aarus\Desktop\Lab9> cd "c:\Users\Aarus\Desktop\Lab9\"
ABCD
EFGH
IJKL
MNOP
QRST
Transposed Vector:
AEIMQ
BFJNR
CGKOS
DHLPT
PS C:\Users\Aarus\Desktop\Lab9>
```

Exercise D




```
//ENSF 337 Lab 9- Ex.D
//Filename- lab9ExD.cpp
//completed by: Aarushi Roy Choudhury

#include <iostream>
#include<cstring>
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
*/
```

```
char convertLower(char c)
{
        if (c >= 'a' && c <= 'z') return c;
        return 'a' + c - 'A';
}
int comparechar(const char* A, const char* B)
{
        int n = strlen(A);
        int m = strlen(B);
        for (int i = 0; i < min(n, m); i++)
        {
                char a = convertLower(A[i]), b = convertLower(B[i]);
                if (a < b) return -1;
                if (b < a) return 1;
        }
        if (n == m) return 0;
        if (n < m) return -1;
        return 1;

}

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(const char** str_array, int n)
{
int i;
int j;
const char* value_to_insert;

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

j = i;
while ( j > 0 && comparechar(str_array[j - 1], value_to_insert) > 0 )
    {
```

```
    str_array[j] = str_array[j - 1];
    j--;
    }

str_array[j] = value_to_insert;

}

}


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;
        }
}
```

```
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:
almond
apple
Blue
cup
Law
nut
pink
Red
white

PS C:\Users\Aarus\Desktop\Lab9>
```

Exercise E

```
#include "Matrix.h"

Matrix::Matrix(int r, int c) : rowsM(r), colsM(c) {

    matrixM = new double *[rowsM];

    assert(matrixM != nullptr);

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

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

}


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) {
        double rowSum = 0.0;
        for (int j = 0; j < colsM; ++j) {
            // adding along the rows
            rowSum += matrixM[i][j];
        }
        sum_rowsM[i] = rowSum;
    }

}

void Matrix::sum_of_cols() const {

    for (int i = 0; i < colsM; ++i) {
        double colSum = 0.0;
        for (int j = 0; j < rowsM; ++j) {
            colSum += matrixM[j][i];
        }
        sum_colsM[i] = colSum;
    }
}

void Matrix::copy(const Matrix &source) {

    if (source.matrixM == nullptr) {

        matrixM = nullptr;
        sum_rowsM = nullptr;
```

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

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

    sum_rowsM = new double[rowsM];

    assert(sum_rowsM != nullptr);

    sum_colsM = new double[colsM];

    assert(sum_colsM != nullptr);

    matrixM = new double *[rowsM];

    assert(matrixM != nullptr);

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

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

    }

    for (int i = 0; i < rowsM; ++i) {
        sum_rowsM[i] = source.sum_rowsM[i];
    }

    for (int i = 0; i < colsM; ++i) {
        sum_colsM[i] = source.sum_colsM[i];
    }


}
```

```
void Matrix::destroy() {

    delete[] sum_colsM;
    delete[] sum_rowsM;

    for (int i = 0; i < rowsM; i += 1) {
        delete[] matrixM[i];
    }
    delete[] matrixM;


}
```

```
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 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 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
```