Lab Assignment #9 Report

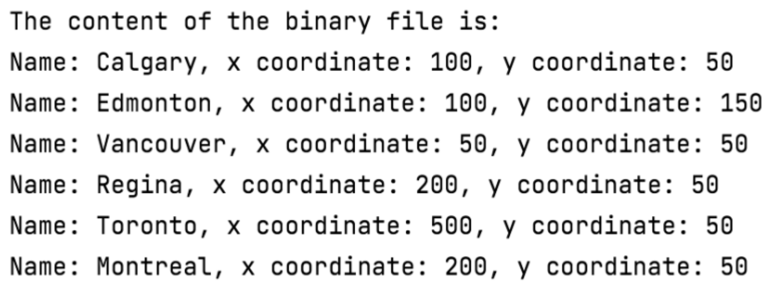
Course: ENSF 337 – Programming Fundamentals for Software and Computer
Lab #: 9
Instructor: Dr. Maan Khedr
Student Name: Sadia Khandaker
Lab Section: B04
Submission Date: December 3, 2021

Exercise B: C++ File I/O

Program:

```
/* File Name: lab9ExB.cpp
 * Assignment: Lab 9 Exercise B
 * Lab Section: B04
 * Completed by: Sadia Khandaker
 * Submission Date: December 3, 2021
 */

#include <iostream>
#include <fstream>
#include <sstream>
#include <stdlib.h>

using namespace std;
struct City {
    double x, y;
    char name[30];
};

void write_binary_file(City cities[], int size, char* filename);
/* PROMISES: attaches an ofstream object to a binary file named "filename"
and
 * writes the content of the array cities into the file.
 */

void print_from_binary(char* filename);
/* PROMISES: attaches an ifstream object to a binary file named "filename"
and
 * reads the content of the file (one record at a time and displays it on the
 * screen.
 */

int main() {
    const int size = 6;
    char bin_filename[] = "cities.bin";

    City cities[size] = {{100, 50, "Calgary"},
                         {100, 150, "Edmonton"},
                         {50, 50, "Vancouver"},
                         {200, 50, "Regina"},
                         {500, 50, "Toronto"},
                         {200, 50, "Montreal"}};

    write_binary_file(cities, size, bin_filename);
    cout << "\nThe content of the binary file is:" << endl;
    print_from_binary(bin_filename);
    return 0;
}

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++)
        stream.write((char*)&cities[i], sizeof(City));
    stream.close();
}

void print_from_binary(char* filename) {
    ifstream inObj(filename, ios::out | ios::binary);
    if(inObj.fail()) {
        cerr << "Failed to open file: " << filename << endl;
        exit(1);
    }
    const int size = 6;
    City cities[size];
    for(int i =0; i < size; i++) {
        inObj.read(reinterpret_cast<char *>(&cities[i]), sizeof(City));
    }
    for (int i = 0; i < size; i++) {
        cout << "Name: " << cities[i].name <<", x coordinate: "<< cities[i].x
<<", y coordinate: "<< cities[i].y << endl;
    }
    inObj.close();
}
```

Output:



```
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
```

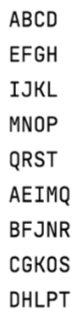
Exercise C: Using C++ library classes, vector and string

Program:

```
/* File Name: lab9ExC.cpp
 * Assignment: Lab 9 Exercise C
 * Completed by: Sadia Khandaker
 * Submission Date: December 3, 2021
 */

#include<vector>
#include<string>
#include <iostream>
using std::cout;
using std::cerr;
using std::endl;
using std::vector;
using std::string;

typedef vector<string> String_Vector;

String_Vector transpose(const String_Vector& sv);
// REQUIRES:
//    sv.size() >= 1
//    All the strings in sv are the same length, and that length is >= 1.
// PROMISES:
//    Return value is the "transpose" of sv, as defined in the Exercise B
//    instructions.

int main() {

    const int ROWS = 5;
    const int COLS = 4;

    char c = 'A';
    String_Vector sv;
    sv.resize(ROWS);

    for(int i = 0; i < ROWS; i++)
        for(int j = 0; j < COLS; j++) {
            sv.at(i).push_back(c);
            c++;
            if(c == 'Z' + 1)
                c = 'a';
            else if (c == 'z' + 1)
                c = 'A';
        }


    for(int i = 0; i < ROWS; i++) {
        cout<< sv.at(i);
        cout << endl;
    }

    String_Vector vs = transpose(sv);
    for(int i = 0; i < (int)vs.size(); i++)
```

```
        cout << vs.at(i) << endl;

    return 0;
}



String_Vector transpose (const String_Vector& sv) {

    int ROWS = (int)sv.at(0).size();
    int COLS = (int)sv.size();
    String_Vector vs;
    vs.resize(ROWS);

    for (int i = 0; i < ROWS; i++) {
        for (int j = 0; j < COLS; j++) {
            vs.at(i).push_back(sv.at(j).at(i));
        }
    }
    return vs;
}
```

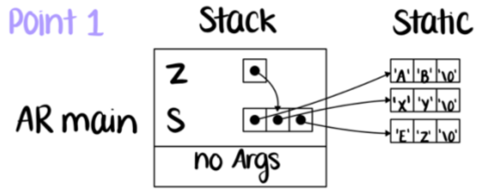
Output:



```
ABCD
EFGH
IJKL
MNOP
QRST
AEIMQ
BFJNR
CGKOS
DHLPT
```

Exercise D: Working with Array of Pointers

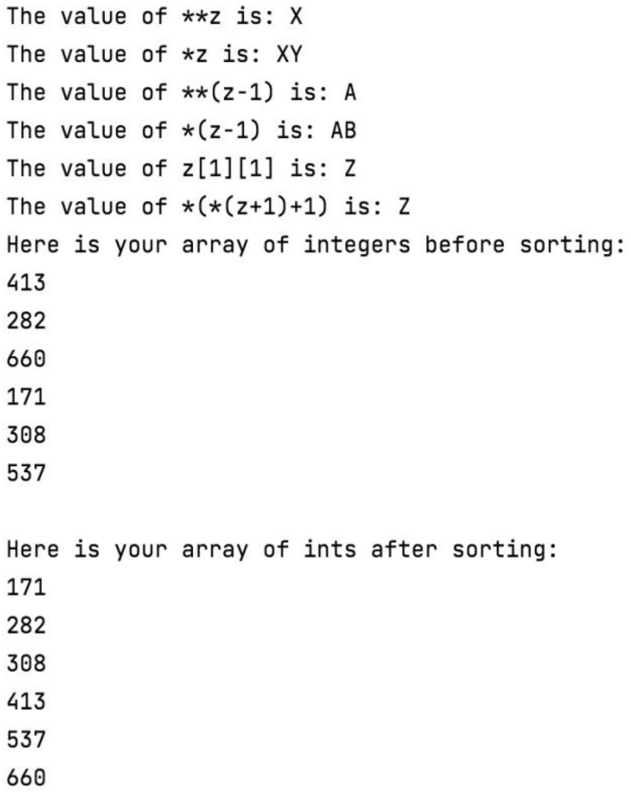
AR Diagram:



Program:

```
/* File Name: lab9ExD.cpp
 * Assignment: Lab 9 Exercise D
 * Lab Section: B04
 * Completed by: Sadia Khandaker
 * Submission Date: December 3, 2021
 */

#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];

        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) {
    const char* tmp;
    for(int i = 0; i<n; i++) {
        tmp = str_array[i];
        for(int j = i - 1; j >= 0; --j){
            if(strcmp(tmp, str_array[j]) == 1) {
                str_array[j+1] = str_array[j];
            }
            else {
                str_array[j+1] = tmp;
                break;
            }
        }

    }
}
```

Output:



```
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
```

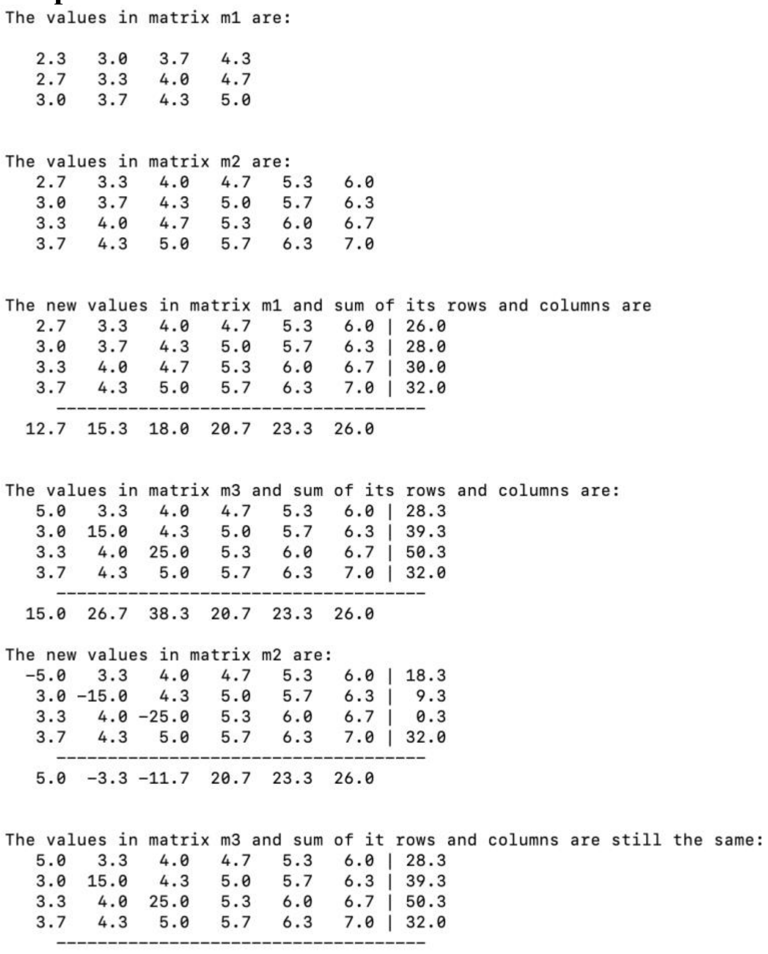
```
Here is your array of strings before sorting:
Red
Blue
pink
apple
almond
white
nut
Law
cup

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

Exercise E: Pointer-to-Pointers and Command-line Arguments

Program:

```
/* File Name: matrix.cpp
 * Assignment: Lab Assignment 8 Exercise E
 * Lab Section: B04
 * Completed by: Sadia Khandaker
 * Submission Date: December 3, 2021
 */

#include "matrix.h"

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

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

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

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

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

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

    return *this;
}

double Matrix::get_sum_col(int i) const
{
    assert(i >= 0 && i < colsM);
    return sum_colsM[i];
}

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


void Matrix::sum_of_rows()const {
    double sum;
    for (int i = 0; i < rowsM; i++) {
        sum = 0;
        for (int j = 0; j < colsM; j++) {
            sum += matrixM[i][j];
            sum_rowsM[i]=sum;
        }
    }
    //cout << "\nSorry I don't know how to calculate sum of rowsM in a
matrix. ";
}

void Matrix::sum_of_cols()const {
    double sum;
    for (int i = 0; i < colsM; i++) {
        sum = 0;
        for (int j = 0; j < rowsM; j++) {
            sum += matrixM[j][i];
            sum_colsM[i] =sum;
        }
    }
    //cout << "\nSorry I don't know how to calculate sum of columns in a
matrix. ";
}

void Matrix::copy(const Matrix& source)
{
    if(source.matrixM == NULL){
        matrixM = NULL;
        sum_rowsM = NULL;
        sum_colsM = NULL;
        rowsM = 0;
        colsM = 0;
        return;
    }

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

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

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

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

    for (int i = 0; i < rowsM; i++){
        matrixM[i] = new double [colsM];
        }
    for(int i = 0; i < rowsM; i++) {
        for (int j = 0; j < colsM; j++)
            matrixM[i][j] = source.matrixM[i][j];
    }
    for (int i = 0; i < colsM; i++){
        sum_colsM[i] = source.sum_colsM[i];
        }
    for (int i = 0; i < rowsM; i++) {
        sum_rowsM[i] = source.sum_rowsM[i];
        }
    //cout << "\nSorry copy fucntion is defective. ";
    }

void Matrix::destroy()
{
    delete sum_rowsM;
    delete sum_colsM;
    delete *matrixM;
    delete matrixM;
    //cout << "\nProgram ended without destroying matrices.\n";
}
```

Output:



```
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
     ------------------------------------
```