**Course: Programming Fundamental – ENSF 337**

**Lab #: Lab 9**

**Instructor: M. Moussavi**

**Student Name: Michael Jeremy Olea (30045602)**

**Lab Section: B01**

**Date submitted: Nov. 28, 2018**

**Exercise B:**

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

/\* Studnets must complete the implementaiton of this file. \*/

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

if(input.fail())

{

cout << "Error: Cannot open" << filename << endl;

exit(1);

}

*int* i = 0;

while(i == 0)

{

City x;

input.read((*char*\*)&x,sizeof(City));

if(input.eof())

i=1;

if(i==0)

{

cout <<"Name: " << x.name << ", ";

cout <<"x coordinate: " << x.x << ", ";

cout <<"y coordinate: " << x.y << " \n";

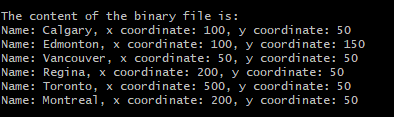
}

}

input.close();

}

Output:



**Exercise C:**

String\_Vector transpose (const String\_Vector& sv) {

// STUDENTS MUST COMPLETE THE DEFINITION OF THIS FUNCTION.

*int* size = (*int*)sv.at(0).size();

String\_Vector vs(size);

cout << "\n"; //just to put a space when printing

for(*int* i=0; i<(*int*)vs.size();i++)

{

vs.at(i).resize((*int*)sv.size());

}

for( *int* i = 0; i<(*int*)sv.size();i++)

{

for(*int* j = 0;j<(*int*)sv.at(i).size();j++)

{

vs.at(j).at(i) = sv.at(i).at(j);

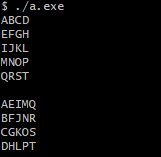
}

}

return vs;

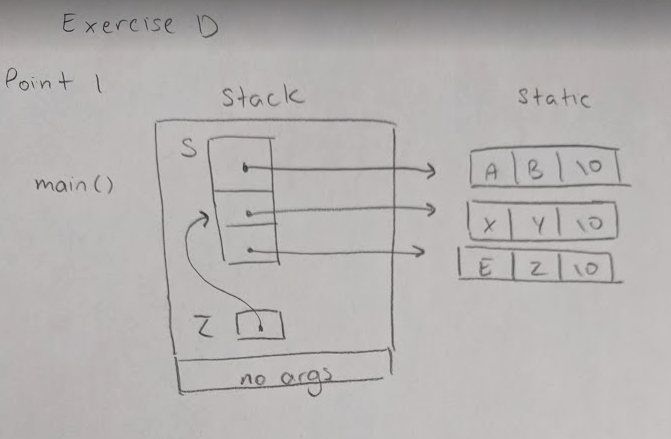
}

Output:



Exercise D:

AR-Diagram:



#include <iostream>

using *namespace* std;

#include <string.h>

*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)

{

*int* i;

*int* j;

const *char* \*select;

for(i = 1; i < n; i++)

{

select = str\_array[i];

j = i;

while(j > 0 && strcmp(str\_array[j-1],select) > 0)

{

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

j--;

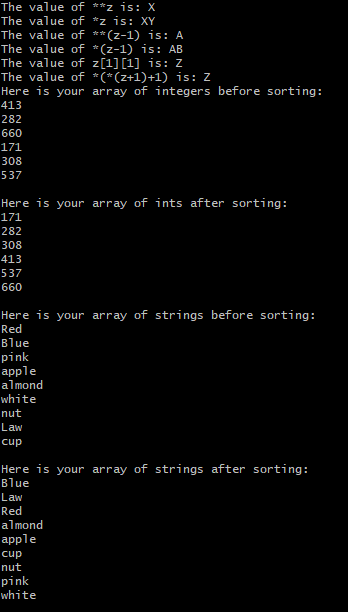
}

str\_array[j] = select;

}

}

Output:



Exercise E:

#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++)

{

*double* sum = 0;

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

{

sum += matrixM[i][j];

}

sum\_rowsM[i] = sum;

}

}

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

{

for(*int* i = 0; i < colsM; i++)

{

*double* sum = 0;

for(*int* j = 0; j < rowsM; j++)

{

sum += matrixM[j][i];

}

sum\_colsM[i] = sum;

}

}

*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];

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\_of\_rows();

sum\_of\_cols();

}

*void* Matrix::destroy()

{

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

{

delete[] matrixM[i];

}

delete[] matrixM;

delete[] sum\_rowsM;

delete[] sum\_colsM;

}

Output:

