|  |
| --- |
| **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;

    }

}

Text

Description automatically generated

**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;

}

Graphical user interface, text

Description automatically generated

**Exercise D**

Diagram, engineering drawing

Description automatically generated

//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;

        }

}

Text

Description automatically generated

**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;

}

Calendar

Description automatically generated



