Programming Fundamentals – ENSF 337

Lab 9

M. Moussavi

Jaxon Braun

B01

Submitted on December 3, 2021

Exercise B: C++ FileI\O

```
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 cities[size];
    for (int i = 0; i < size; i++){
        stream.read(reinterpret_cast<char*>(&cities[i]), sizeof(City));
    }
    for (int i = 0; i < size; i++){
        cout << "Name: " << cities[i].name;
        cout << "\tx-coordinate: " << cities[i].x;
        cout << "\ty-coordinate: " << cities[i].y << endl;
    }
    stream.close();
}
```

```
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

String_Vector transpose (const String_Vector& sv) {

```
    String_Vector vs;
    int cols = sv.size();
    int rows = sv.at(0).size();

    vs.resize(rows);
    for (int j = 0; j < rows; j++)
        vs.at(j).resize(cols);

    for (int i = 0; i < rows; i++){
        for (int j = 0; j < cols; j++){
            vs.at(i).at(j) = sv.at(j).at(i);
        }
    }
    return vs;
}
```

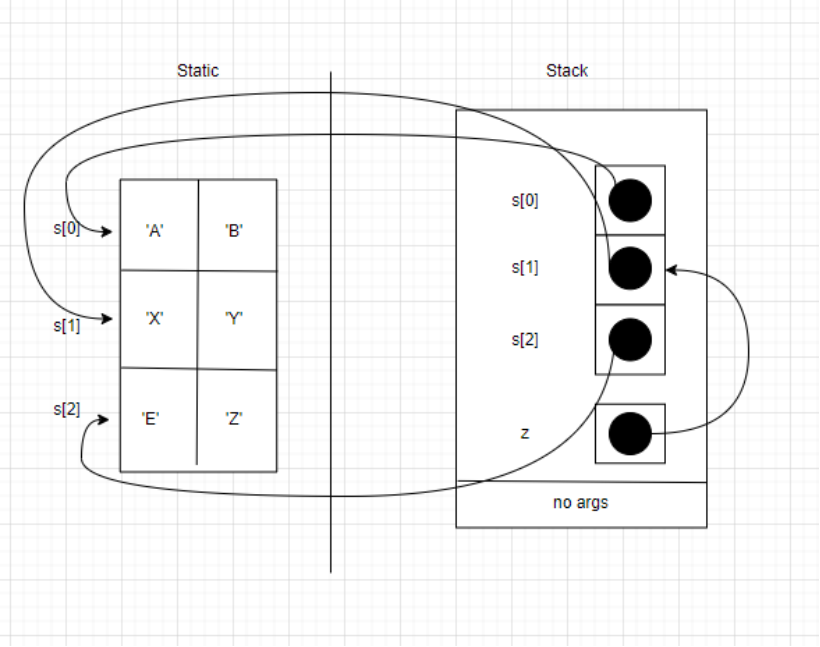
```
ABCD
EFGH
IJKL
MNOP
QRST

AEIMQ
BFJNR
CGKOS
DHLPT
```

```
ABCD
EFGH
IJKL
MNOP
QRST
UVWX
YZab
cdef
ghij
klmn

AEIMQUYcgk
BFJNRVZdhl
CGKOSWaeim
DHLPTXbfjn
```

Exercise D: Working with Array of Pointers




```
#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;
    cout << z << 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, n_elements);


    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){
```

```
    for (int i = 0; i < n - 1; i++){
        for (int j = i + 1; j < n; j++){
            if (str_array[i] > str_array[j]){
                const char* temp = str_array[i];
                str_array[i] = str_array[j];
                str_array[j] = temp;
            }
        }
    }
}
```

```
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
0x61fefc
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
```

Could not get array to sort, followed the exact procedure shown in the notes, I don't understand what is wrong with my implementation of the function

Exercise E: Pointer-To-Pointer and Command-line Arguments

```
//   matrix.cpp



#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;
}
```

```
PS C:\Users\Jaxon Braun\Documents\UofC\Fall 2021\ENSF 337\lab9> ./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
```