|  |
| --- |
| **Course: Programming Fundamental – ENSF 337**  Lab #: 9  Instructor: Khedr  Student Name: Aleksander Berezowski  Lab Section: B04  Date submitted: On or before December 3rd |

**Exercise B**

*void* print\_from\_binary(*char*\* *filename*) {  
 *ifstream* stream(*filename*, *ios*::out **|** *ios*::binary);  
 *if*(stream.fail()){  
 cout **<<** *filename* **<<** " failed to open" **<<** endl;  
 exit(1);  
 }  
 City cityObjects [1000];  
 stream.seekg(0L, *ios*::end);  
 *int* amount = stream.tellg()/*sizeof*(City);  
 stream.seekg(0L, *ios*::beg);  
 stream.read((*char*\*)cityObjects, *sizeof*(City) \* amount);  
 stream.close();  
  
 *int* i = 0;  
 *while* (i<amount){  
 cout **<<** "Name: " **<<** cityObjects[i].name;  
 cout **<<** ", X Coordinate:" **<<** cityObjects[i].x;  
 cout **<<** ", Y Coordinate:" **<<** cityObjects[i].y **<<** endl;  
 i++;  
 }  
}

Text

Description automatically generated

**Exercise C**

*String\_Vector* transpose (*const String\_Vector*& *sv*) {  
 *String\_Vector* vs;  
 vs.resize(*sv***[**0**]**.size());  
 *int* x, y;  
  
 *for* (x = 0; x < *sv***[**x**]**.size(); x++){  
 *for* (y = 0; y < *sv*.size(); y++){  
 vs**[**x**]**.push\_back(*sv***[**y**][**x**]**);  
 }  
 }  
 *return* vs;  
}

Text

Description automatically generated

**Exercise D**

Diagram

Description automatically generated

Predicted 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

#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];  
   
 */\* 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, j;  
 *const char* \*t;  
 *for* (i = 0; i < *n*; i++) {  
 *for* (j = i + 1; j < *n*; j++) {  
 *if* (\*(*str\_array* + j) < \*(*str\_array* + i)) {  
 t = \*(*str\_array* + i);  
 \*(*str\_array* + i) = \*(*str\_array* + j);  
 \*(*str\_array* + j) = t;  
 } *else if* (\*(*str\_array* + j)[0] == \*(*str\_array* + i)[0]) {  
 *if* (\*(*str\_array* + j)+1 < \*(*str\_array* + i)+1) {  
 t = \*(*str\_array* + i);  
 \*(*str\_array* + i) = \*(*str\_array* + j);  
 \*(*str\_array* + j) = t;  
 }  
 }  
 }  
 }  
  
}

Text

Description automatically generatedText

Description automatically generated

**Exercise E**

*// 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*{  
 *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;  
 }  
}  
  
*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;  
 }  
}  
  
*void* Matrix::copy(*const* Matrix& *source*)  
{  
 *// THIS FUNCITON IS DEFECTIVE AND DOSEN'T PROPERLY MAKE THE COPY OF SROUCE  
 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*.at(i,j);  
 }  
 sum\_rowsM[i] = *source*.sum\_rowsM[i];  
 }  
 *for* (*int* i = 0; i < colsM; ++i) {  
 sum\_colsM[i] = *source*.sum\_colsM[i];  
 }  
}  
  
   
  
*void* Matrix::destroy()  
{  
 *for*(*int* i = 0; i<rowsM; i++) {  
 *delete* matrixM[i];  
 }  
 *delete* matrixM;  
 *delete* sum\_colsM;  
 *delete* sum\_rowsM;  
}

Calendar

Description automatically generated

Calendar

Description automatically generated