#include <iostream>

using namespace std;

struct matrix {

int rows, cols;

double \* arrayptr;

void print() {

cout << "rows = " << rows << " cols = " << cols << "\n";

}

matrix(int r, int c) {

rows = r;

cols = c;

arrayptr = new double[rows\*cols];

}

~matrix();

double get(int a, int b);

void set(int row, int col, double value);

void resize(int rsize, int csize);

matrix clone();

};

matrix::~matrix() {

delete [] arrayptr;

arrayptr = NULL;

}

void matrix::resize(int rsize, int csize) {

cout << "resizing...\n";

int rowlim, collim;

if (rsize<0 || csize<0) {

cerr << "Error in resize\n";

return;

}

if (rsize<rows || csize<cols) {

cout << "Warning: New matrix size is smaller than current matrix. Continue? (Y or N)\n";

char answer;

cin >> answer;

if (answer == 'N') return;

}

else {

if (rsize>rows) rowlim = rows;

if (rsize<rows || rsize==rows) rowlim = rsize;

if (csize>cols) collim = cols;

if (csize<cols || csize==cols) collim = csize;

double \* newarray;

newarray = new double[rsize\*csize];

for (int i=0; i<rowlim; i++) {

for (int j=0; j<collim; j++)

newarray[i\*csize+j]=arrayptr[i\*cols+j];

}

delete [] arrayptr;

arrayptr = newarray;

rows = rsize;

cols = csize;

cout << "all done\n";

}

}

void matrix::set(int row, int col, double value) {

if (row<0 || col<0) {

cerr << "Error in set\n";

return;

}

if (row>rows && col>cols) resize(row+1,col+1);

else if (row>rows) resize(row+1,cols);

else if (col>cols) resize(rows,col+1);

arrayptr[row\*cols+col] = value;

}

double matrix::get(int a, int b) {

if (a<0 || a>=rows || b<0 || b>=cols) {

cerr << "Error in get\n";

return NULL;

}

return arrayptr[a\*cols+b];

}

matrix matrix::clone() {

matrix arrayclone(rows,cols);

arrayclone.arrayptr = new double[rows\*cols];

for (int i=0; i<rows\*cols; i++) {

arrayclone.arrayptr[i] = arrayptr[i];

}

return arrayclone;

}

matrix add(matrix a, matrix b) {

if (a.rows != b.rows || a.cols != b.cols) {

cerr << "Error in add\n";

exit(1);

}

matrix result(a.rows,a.cols);

result.arrayptr = new double[a.rows\*a.cols];

for (int i=0; i<a.rows\*a.cols; i++) {

result.arrayptr[i] = a.arrayptr[i] + b.arrayptr[i];

}

return result;

}

matrix subtract(matrix a, matrix b) {

if (a.rows != b.rows || a.cols != b.cols) {

cerr << "Error in subtract\n";

exit(1);

}

matrix result(a.rows,a.cols);

result.arrayptr = new double[a.rows\*a.cols];

for (int i=0; i<a.rows\*a.cols; i++) {

result.arrayptr[i] = a.arrayptr[i] - b.arrayptr[i];

}

return result;

}

matrix \*multiply(matrix &a, matrix &b) {

if (a.cols != b.rows) {

cerr << "Error in multiply\n";

return NULL;

}

matrix \*result = new matrix(a.rows,b.cols);

result->arrayptr = new double[a.rows\*b.cols];

double \*sum = new double[a.rows\*b.cols];

for (int i=0; i<a.rows; i++) {

for (int j=0; j<b.cols; j++) {

for (int k=0; k<b.rows; k++) {

sum[i\*b.cols+j] += a.arrayptr[i\*a.cols+k]\*b.arrayptr[k\*b.cols+j];

result->arrayptr[i\*b.cols+j] = sum[i\*b.cols+j];

}

}

}

return result;

}

void transpose(matrix &a) {

double \*result = new double[a.rows\*a.cols];

for (int i=0; i<a.rows; i++) {

for (int j=0; j<a.cols; j++)

result[j\*a.rows+i] = a.arrayptr[i\*a.cols+j];

}

delete [] a.arrayptr;

a.arrayptr = result;

int temp = a.rows;

a.rows = a.cols;

a.cols = temp;

}

void main() {

matrix a(2,2);

matrix b(2,2);

a.set(0,0,111);

a.set(0,1,222);

a.set(1,0,333);

a.set(1,1,444);

b.set(0,0,151);

b.set(0,1,575);

b.set(1,0,252);

b.set(1,1,797);

matrix c = add(a,b);

cout << "Added value at [0][0] = " << c.get(0,0) << "\n";

cout << "Added value at [1][0] = " << c.get(1,0) << "\n";

cout << "Added value at [1][1] = " << c.get(1,1) << "\n\n";

matrix d = subtract(a,b);

cout << "Subtracted value at [0][0] = " << d.get(0,0) << "\n";

cout << "Subtracted value at [1][0] = " << d.get(1,0) << "\n";

cout << "Subtracted value at [1][1] = " << d.get(1,1) << "\n\n";

matrix x(3,4);

matrix y(4,2);

x.set(0,0,1);

x.set(0,1,2);

x.set(0,2,3);

x.set(0,3,4);

x.set(1,0,5);

x.set(1,1,6);

x.set(1,2,7);

x.set(1,3,8);

x.set(2,0,0);

x.set(2,1,1);

x.set(2,2,0);

x.set(2,3,1);

y.set(0,0,3);

y.set(0,1,6);

y.set(1,0,1);

y.set(1,1,2);

y.set(2,0,4);

y.set(2,1,1);

y.set(3,0,1);

y.set(3,1,0);

matrix z = \*multiply(x,y);

cout << "Multiplied value at [0][0] = " << z.get(0,0) << "\n";

cout << "Multiplied value at [0][1] = " << z.get(0,1) << "\n";

cout << "Multiplied value at [1][0] = " << z.get(1,0) << "\n";

cout << "Multiplied value at [1][1] = " << z.get(1,1) << "\n";

cout << "Multiplied value at [2][0] = " << z.get(2,0) << "\n";

cout << "Multiplied value at [2][1] = " << z.get(2,1) << "\n\n";

transpose(y);

cout << "Transposed value at [0][0] = " << y.get(0,0) << "\n";

cout << "Transposed value at [0][1] = " << y.get(0,1) << "\n";

cout << "Transposed value at [0][2] = " << y.get(0,2) << "\n";

cout << "Transposed value at [0][3] = " << y.get(0,3) << "\n";

cout << "Transposed value at [1][0] = " << y.get(1,0) << "\n";

cout << "Transposed value at [1][1] = " << y.get(1,1) << "\n";

cout << "Transposed value at [1][2] = " << y.get(1,2) << "\n";

cout << "Transposed value at [1][3] = " << y.get(1,3) << "\n";

}

**Output:**

