Effective and Modern C++ Programming

Lab 1 – Copy and Move Semantics

Exercise 1. Copy semantics

Implement a class Matrix that will store a two dimensional matrix (N x M) with double coefficients. Coefficients should be stored in dynamically allocated one dimensional array (referred as data).

Provide the following functionalities:

```
Matrix m1;         // default constructor: N=M=0, data=nullptr
Matrix m2(3,4);    // N = 3, M = 4, data = N*M doubles set to 0.0
Matrix m3({{1,2,3},{32, 23, 22},{3,234,23,44}}); // from initializer_list
cout << m2(1,1) << endl;    // 0  - the first element in data (we count from 1)
cout << m3(2,2) << endl;    // 23
cout << m3;                 // prints matrix

cout << "Copy semantics \n";
Matrix m4 = m2;     // copy constructor that makes deep copy of the matrix m2
m4 = m3;            // copy assignment makes deep copy of the matrix m3
```

- Constructor with initializer_list parameter should construct the smallest matrix NxM that can store provided data. If in given row there is not enough data initialize remaining elements to zero.
- Provide also destructor that frees allocated memory.
- All constructors should print short debug information that they was called, similar to the following output:

```
 constructor of 3x4 matrix
 constructor of 3x4 matrix from initializer_list
0
23
{1, 2, 3, 0}
{32, 23, 22, 0}
{3, 234, 23, 44}
Copy semantics
 copy constructor
 copy assignment operator
```

Exercise 2. Move semantics
Implement move constructor and move assignment operator for class Matrix.
Instead of making a copy of data they should just move data pointer from the source object to the destination object.
Make sure to leave source object in the correct state and that there are no leaks of resources.

Implement unary operator - that for given matrix A returns matrix -A.

Observe copy elision that omit copy and move constructors and resulting in zero-copy pass-by-value semantics.

Example

```
cout << "Move semantics \n";
Matrix m7 = std::move(m2);
m4 = -m3;
cout << "Copy elision \n";
Matrix m6 = -m4;
Matrix * pm = new Matrix(-m4);
cout << m6(2,1) << endl; // 32
```

Expected output:

```
Move semantics
 move constructor
 constructor of 3x4 matrix
 move assignment operator
Copy elision
 constructor of 3x4 matrix
 constructor of 3x4 matrix
```

Exercise 3. Inheritance of constructors

Implement class MatrixWithLabel that is a subclass of Matix and add field that stores matrix label (as string, default value "A"). Class MatrixWithLabel should inherit all constructors from Matrix (not reimplement them!) and add two new constructors:

```
MatrixWithLabel(label, numberOfRows, numberOfColumns)

MatrixWithLabel(label, initializer_list<...>)
```

Add also getLabel and setLabel methods.

Example

```
cout << "Inheritance \n";
MatrixWithLabel l0("B", 3, 4);
MatrixWithLabel l1({{1,2},{4,5}});
l1.setLabel("A");
MatrixWithLabel l2 = l1;
MatrixWithLabel l3 = std::move(l1);
cout << l2.getLabel() << " " << l3.getLabel() << endl;
//    cout << l1.getLabel() << endl;
```

Expected output:

```
Inheritance
 constructor of 3x4 matrix
 constructor of 2x2 matrix from initializer_list
 copy constructor
 move constructor
A A
```

- Check if default copy an move constructors work as expected.
- Implement your own copy constructor. Observe what happened to default move constructor.
- Explicitly add default move constructors and assignment operators.