Valentin Helml
Dirk Praetorius
Julian Streitberger

Summer term 2022
20 May 2022

Exercise
Scientific programming in mathematics

Series 9

Exercise 9.1. Write a class `Matrix` to save $n \times n$ square matrices. The class contains the data members `n (int)` for the dimension, a dynamic vector of type `vector <double>` and a type `type (char)`. The type allows to distinguish between fully populated matrices (type `'F'`), lower triangular matrices (type `'L'`), and upper triangular matrices (type `'U'`). A fully populated matrix should be stored in Fortran-style, i.e., the coefficients are stored columnwise in a vector with n^2 entries. The coefficients of a (lower or upper) triangular matrix should be stored in a vector with $\sum_{j=1}^{n} j = n(n+1)/2$ entries. Implement the following functionalities:

- The standard access methods to work with the class, where access to A_{jk} (for admissible indices j and k) should be provided via `A(j,k)`.
- The standard constructor, which allocates a 0×0 matrix of type `'F'`.
- A constructor, which gets the dimension and type as input parameters and allocates the corresponding matrix with all entries initialized with 0.
- A constructor, which gets the dimension, type, and a value as input parameters, allocates the corresponding matrix, and initializes all entries with the given value.
- Why is it not necessary to write destructor, copy constructor, and assignment operator?

Note that the implementation of the access methods depends on the type of the matrix. Test your implementation with suitable examples.

Exercise 9.2. Extend the class `Matrix` from the previous Exercise 9.1 by the capability of solving the linear system of equation $Ax = b$ for upper triangular matrices, i.e., matrices of type `'U'` with the vector template from the lecture. More precisely, overload the operator `|` so that typing `x=U | b` for an upper triangular matrix $U \in \mathbb{R}^{n\times n}$, and a vector $b \in \mathbb{R}^n$ of type `Vector` computes and returns the solution $x \in \mathbb{R}^n$ of the system $Ux = b$ of type `Vector` from the lecture. Use `assert` to check that the dimensions match and that $U_{jj} \neq 0$ for all $j = 1, \dots, n$. Test your implementation appropriately.

Exercise 9.3. Extend the class `Matrix` from Exercise 9.1 by the following methods:

- `double columnSumNorm()`, which computes and returns the column sum norm defined by

$$\|A\|_1 := \max_{k=0,\dots,n-1} \sum_{j=0}^{n-1} |a_{jk}|;$$

- `double rowSumNorm()`, which computes and returns the row sum norm defined by

$$\|A\|_\infty := \max_{j=0,\dots,n-1} \sum_{k=0}^{n-1} |a_{jk}|;$$

- `double frobeniusNorm()`, which computes and returns the Frobenius norm defined by

$$\|A\|_F := \left(\sum_{j,k=0}^{n-1} |a_{jk}|^2 \right)^{1/2}.$$

Note that for lower (resp. upper) triangular matrices, the methods should access only coefficients a_{jk} (resp. a_{kj}) with $0 \le k \le j \le n-1$. Test your implementation with suitable examples!

Exercise 9.4. A matrix $A \in \mathbb{R}^{n \times n}$ is called diagonal if $a_{jk} = 0$ for all $j \neq k = 0, \dots, n-1$, symmetric if $A^\top = A$, and skew-symmetric if $A^\top = -A$. Extend the class `Matrix` from Exercise 9.1 by the following methods:

- `double trace()`, which computes and returns the trace defined by

$$\mathrm{tr}(A) = \sum_{j=0}^{n-1} a_{jj};$$

- `bool isDiagonal()`, which checks whether a matrix is diagonal;
- `bool isSymmetric()`, which checks whether a matrix is symmetric;
- `bool isSkewSymmetric()`, which checks whether a matrix is skew-symmetric.

Note that for lower (resp. upper) triangular matrices, the methods should access only coefficients a_{jk} (resp. a_{kj}) with $0 \le k \le j \le n-1$. What do symmetric triangular matrices look like? What do skew-symmetric triangular matrices look like? Test your implementation with suitable examples.

Exercise 9.5. Extend the class `Matrix` from Exercise 9.1 by a constructor, which creates a matrix of given dimension and type with random entries. To this end, in addition to the desired dimension and matrix type, the constructor receives two parameters `lb` $\le$ `ub` of type `double`, which define a lower bound and an upper bound for the random entries of the matrix, respectively. This means that the random entries $(a_{ij})_{i,j=1,\dots,n}$ of the matrix fulfill the inequalities

- Type `'F'`: `lb` $\le a_{ij} \le$`ub` for $0 \le i, j \le n-1$,
- Type `'L'`: `lb` $\le a_{ij} \le$`ub` for $0 \le j \le i \le n-1$,
- Type `'U'`: `lb` $\le a_{ij} \le$`ub` for $0 \le i \le j \le n-1$.

Hint: One can generate (pseudo-)random numbers between 0 and 1 by means of

```
srand(time(NULL));
double rnd = (double) rand() / RAND_MAX;
```

using the libraries `ctime` and `cstdlib`.

Exercise 9.6. Implement the simple *tic-tac-toe* game. You can find the rules at tictactoe. Your program should fulfill the following criteria:

1. Two players play against each other.
2. The play field should be printed to the screen after each move.
3. After each move, the program should check whether one of the players has won.
4. If the play field is full and none of the players has won, the message `'Tied game'` should be printed to the screen.

Your implementation should use the class `Matrix` and its functionalities from Exercise 9.1, e.g., the play field should be stored in a 3×3 fully populated matrix. Each player should have its own character symbol, e.g., `'1'` for player 1 and `'2'` for player 2. Challenge your friends to test your implementation!

Exercise 9.7. The private members of a class can only be accessed indirectly via access methods. What is the output of the following C++ program? Why is this possible? Explain why this is a bad programming style.

```
#include <iostream>
using std::cout;
using std::endl;

class Test{

        private:
        int N;

        public:
        void setN(int N_in) { N = N_in; };
        int getN(){ return N; };
        int* getptrN(){ return &N; };

};

int main(){

        Test A;
        A.setN(5);
        int* ptr = A.getptrN();
        cout << A.getN() << endl;
        *ptr = 10;
        cout << ptr << endl;
        cout << A.getN() << endl;

        return 0;
}
```

Exercise 9.8. Implement a class `Person`, which contains the data members `name` and `address` (of type `string`). Derive from `Person` the class `Student` that contains the additional data fields `studentNumber` (`int`) and `study` (`string`). Derive from `Person` also the class `Employee` that contains the additional data field `salary` (`double`) and `job` (`string`). Write the standard access methods, constructors, and destructors for all classes. Implement the method `print` for the base class `Person`. The method should print the name and address of a person to the screen. Redefine this function for the derived classes `Student` and `Employee` so that also the additional data fields are printed. Test your implementation appropriately.