Dirk Praetorius
Michele Ruggeri

Summer semester 2019
May 10, 2019

Scientific programming in mathematics

Exercise sheet 8

Constructor and destructor

Starting with this exercise sheet, please follow the conventions (for names, variable declarations and files) described in slides 202–211 of the lecture notes.

Exercise 8.1. Write a class `Matrix` to save n-by-n square matrices. The class contains the data members `n` (`int`) for the dimension, a dynamical vector `coeff` (`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 lower triangular matrix L and an upper triangular matrix U have the structure

$$U = \begin{pmatrix} u_{11} & u_{12} & u_{13} & \dots & u_{1n} \\ & u_{22} & u_{23} & \dots & u_{2n} \\ & & u_{33} & \dots & u_{3n} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{nn} \end{pmatrix} \quad \text{and} \quad L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix},$$

respectively, i.e., it holds that $u_{jk} = 0$ if $j > k$ and $\ell_{jk} = 0$ if $j < k$. A fully populated matrix should by stored in Fortran-style, i.e., the coefficients are stored columnwise in a dynamical 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 mutator methods to work with the class;
- The standard constructor, which allocates a 0×0 matrix of type `'F'`;
- A constructor, which gets dimension and type as input parameters and allocates the corresponding matrix with all entries initialized with 0;
- A constructor, which gets dimension, type, and a value as input parameters, allocates the corresponding matrix, and initializes all entries with the given value;
- The destructor.

Note that the implementation of the mutator methods depends on the type of the matrix. A template for the structure of your implementation is provided by the class `Vector` presented in slides 220–222 of the lecture notes. Test your implementation with suitable examples!

Exercise 8.2. Extend the class `Matrix` from Exercise 8.1 by the following methods:

- `void scanMatrix(int n, char type)`, which reads a matrix of given dimension and type from the keyboard;
- `void printMatrix()`, which prints a matrix to the screen.

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 8.3. Extend the class `Matrix` from Exercise 8.1 by the following methods:

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

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

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

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

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

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

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

Exercise 8.4. A n-by-n matrix A is called diagonal if $a_{jk} = 0$ for all $j, k = 0, \ldots, n-1$ with $j \neq k$. A n-by-n matrix A is called symmetric if $A^T = A$, i.e., if $a_{kj} = a_{jk}$ for all $j, k = 0, \ldots, n-1$. A n-by-n matrix A is called skew-symmetric if $A^T = -A$, i.e., if $a_{kj} = -a_{jk}$ for all $j, k = 0, \ldots, n-1$. Extend the class `Matrix` from Exercise 8.1 by the following methods:

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

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

- `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 $0 \leq k \leq j \leq n-1$. How do symmetric triangular matrices look like? How do skew-symmetric triangular matrices look like? Test your implementation with suitable examples!

Exercise 8.5. Extend the class `Matrix` from Exercise 8.1 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 the type of the matrix, the constructor receives two parameters `lb` $\leq$ `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}) of the matrix satisfy the inequalities

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

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 8.6. Implement the simple *tic-tac-toe* game. The rules can be found at

`https://en.wikipedia.org/wiki/Tic-tac-toe`

Your program should fulfill the following criteria:

1. There are two players who 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 8.1, e.g., the play field should be stored in a 3-by-3 fully populated matrix. Each of the player should use its own character symbol, e.g., `'1'` for player 1 and `'2'` for player 2. Challenge your friends to test your implementation!

Exercise 8.7. Write a `Makefile` for the exercises of this sheet which involve the class `Matrix` from Exercise 8.1. It should contain:

- Compilation of all solved exercises.
- Generation of a dynamic library for the matrix class and an example for its usage.

Exercise 8.8. According to the lecture, the private members of a class can only be accessed indirectly via mutator 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;
}
```