Dirk Praetorius,
Michele Ruggeri

Sommersemester 2017
23.05.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 10

Aufgabe 10.1. Laut der Vorlesung ist der Zugriff auf Members einer Klasse vom Typ `private` nur über `set`- und `get`-Methoden der Klasse möglich. Wie lautet die Ausgabe des folgenden C++ Programms? Warum ist das möglich? Erkären Sie, warum das schlechter Programmierstil ist.

```
#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;
}
```

Aufgabe 10.2. Schreiben Sie eine Klasse `SquareMatrix` zur Speicherung quadratischer Matrizen $A \in \mathbb{R}^{n\times n}$. Hierbei sollen die Einträge der Matrix spaltenweise als `double*`, sowie die Größe $n \in \mathbb{N}$ abgespeichert werden. Im Gegensatz zur üblichen Indizierung in C++ soll die Indizierung der Matrixeinträge a_{jk} in Ihrer Klasse `SquareMatrix` von $j, k = 1$ bis n laufen (wie in der Mathematik üblich). Implementieren Sie die folgenden Funktionalitäten: Konstruktor, Destruktor, `set`- und `get`-Methoden für die Matrix-Einträge, und eine `get`-Methode für die Dimension. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen. Speichern Sie den Source-Code unter `squareMatrix.{hpp/cpp}` in das Verzeichnis `serie10`.

Aufgabe 10.3. Eine quadratische Matrix $A \in \mathbb{R}^{n\times n}$ ist

- symmetrisch, falls $a_{jk} = a_{kj}$ für alle $1 \le k, j \le n$ gilt;
- eine untere Dreiecksmatrix, falls alle Einträge oberhalb der Hauptdiagonale null sind, d.h. es gilt $a_{jk} = 0$ für alle $1 \le j < k \le n$;

- eine obere Dreiecksmatrix, falls alle Einträge unterhalb der Hauptdiagonale null sind, d.h. es gilt $a_{jk} = 0$ für alle $1 \leq k < j \leq n$;
- diagonal, falls alle Einträge außerhalb der Hauptdiagonale Null sind, d.h. es gilt $a_{jk} = 0$ für alle $k \neq j$;
- tridiagonal, falls alle Einträge außerhalb der Hauptdiagonale und den beiden ersten Nebendiagonalen Null sind, d.h. es gilt $a_{jk} = 0$ wenn $|k - j| > 1$.

Erweitern Sie die Klasse `SquareMatrix` aus Aufgabe 10.2 um eine Methode `void testMatrix()`, die die Struktur einer Matrix überprüft. Bitte beachten Sie, dass einige Eigenschaften gleichzeitig gelten können, z.B. die Einheitsmatrix erfüllt alle fünf vorgenannten Eigenschaften. Schreiben Sie ein aufrufendes Hauptprogramm, in dem Sie Ihre Implementierung eingehend testen.

Aufgabe 10.4. Eine untere Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ mit

$$L = \begin{pmatrix} \ell_{00} & & & & \mathbf{0} \\ \ell_{10} & \ell_{11} & & & \\ \ell_{20} & \ell_{21} & \ell_{22} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n-1,0} & \ell_{n-1,1} & \ell_{n-1,2} & \dots & \ell_{n-1,n-1} \end{pmatrix}$$

hat höchstens $\frac{n(n+1)}{2} = \sum_{j=1}^{n} j$ nicht-triviale Einträge. Schreiben Sie eine Klasse `matrixL`, in der neben der Dimension $n \in \mathbb{N}$ die Koeffizienten in einem dynamischen Vektor `coeff` der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Speichern Sie L zeilenweise, d.h. der Koeffizient ℓ_{jk} wird in `coeff[j*(j+1)/2+k]` für $0 \leq k \leq j \leq n-1$ gespeichert. Wie kommt man auf diese Formel? Implementieren Sie die folgenden Funktionalitäten: Konstruktor, Destruktor, `set`- und `get`-Methoden für die Matrix-Einträge, und eine `get`-Methode für die Dimension. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen. Speichern Sie den Source-Code unter `matrixL.{hpp/cpp}` in das Verzeichnis `serie10`.

Aufgabe 10.5. Erweitern Sie die Klasse `MatrixL` aus Aufgabe 10.4 um

- eine Methode `scanMatrixL(int n)`, die eine untere Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ von der Tastatur einliest,
- eine Methode `printMatrixL()`, die die Matrix am Bildschirm ausgibt,
- eine Methode `columnsumnormL()`, die die Spaltensummennorm

  $$\|L\| = \max_{k=0,\dots,n-1} \sum_{j=0}^{n-1} |\ell_{jk}|$$

  berechnet und zurückgibt,
- eine Methode `rowsumnormU()`, die die Zeilensummennorm

  $$\|L\| = \max_{j=0,\dots,n-1} \sum_{k=0}^{n-1} |\ell_{jk}|$$

  berechnet und zurückgibt.

Beachten Sie, dass die Methoden nur auf Koeffizienten ℓ_{jk} für $0 \leq k \leq j \leq n-1$ zugreifen können. Schreiben Sie weiters ein aufrufendes main-Programm um Ihre implementierten Methoden zu testen.

Aufgabe 10.6. Schreiben Sie eine Funktion `void MatrixVectorProductU(Vector& b, MatrixL& L, Vector& x)` um das Matrix-Vektor-Produkt $b = Lx \in \mathbb{R}^n$ für eine untere Dreiecksmatrix L und einen Vektor x bei passenden Dimensionen zu berechnen, d.h.

$$b_j = \sum_{k=0}^{n-1} L_{jk} x_k \quad \text{für } j = 0, \dots, n-1. \tag{1}$$

Verwenden Sie die Klasse `MatrixL` aus Aufgabe 10.4 und die Klasse `Vector` aus der Vorlesung. Stellen Sie mittels `assert` sicher, dass die Dimensionen zusammenpassen. Beachten Sie, dass die Funktion nur auf Koeffizienten ℓ_{jk} mit $0 \leq k \leq j \leq n-1$ zugreifen kann. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierungen testen. Speichern Sie den Source-Code unter `multMatrixVectorL.cpp` in das Verzeichnis `serie10`.

Aufgabe 10.7. Beweisen Sie mathematisch mit der Formel des Matrix-Matrix-Produktes

$$C_{ij} = \sum_{k=0}^{n-1} A_{ik} B_{kj} \quad \text{für } i, j = 0, \dots n-1,$$

dass das Produkt $C = AB \in \mathbb{R}^{n \times n}$ zweier unterer Dreiecksmatrizen $A, B \in \mathbb{R}^{n \times n}$ wieder eine untere Dreiecksmatrix ist. Schreiben Sie eine Funktion `void MatrixMatrixProductL(MatrixL& C, MatrixL& A, MatrixL& B)`, um das Matrixprodukt für zwei untere Dreiecksmatrizen bei passenden Dimensionen zu berechnen. Verwenden Sie die Klasse `MatrixL` aus Aufgabe 10.4. Stellen Sie mittels `assert` sicher, dass die Matrizen dieselbe Dimension haben. Beachten Sie, dass die Funktion nur Koeffizienten C_{jk} für $0 \leq k \leq j \leq n-1$ berechnen soll und auch nur auf die entsprechenden Koeffizienten von A und B zugreifen kann. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierungen testen. Speichern Sie den Source-Code unter `multMatrixMatrixL.cpp` in das Verzeichnis `serie10`.

Aufgabe 10.8. Gegeben sei eine untere Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ mit $\ell_{jj} \neq 0$ für alle $j = 0, \dots, n-1$. Zu gegebenem $b \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Lx = b$. Leiten Sie eine Formel her, um die Lösung $x \in \mathbb{R}^n$ von $Lx = b$ zu berechnen, indem Sie die Formel des Matrix-Vektor-Produkts (1) mithilfe der Dreiecksstruktur von L vereinfachen. Implementieren Sie eine Funktion `void solveL(Vector& x, MatrixL& L, Vector& b)`, um für eine untere Dreiecksmatrix $L \in \mathbb{R}^{n \times n}$ und einen Vektor $b \in \mathbb{R}^n$ das System $Lx = b$ zu lösen. Verwenden Sie die Klasse `MatrixL` aus Aufgabe 10.4 und die Klasse `Vector` aus der Vorlesung. Stellen Sie mittels `assert` sicher, dass die Dimensionen zusammenpassen und dass $\ell_{jj} \neq 0$ für alle j gilt. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen. Speichern Sie den Source-Code unter `solveMatrixL.cpp` in das Verzeichnis `serie10`.