

Übungen zur Vorlesung Einführung in das Programmieren für TM

Serie 9

Aufgabe 9.1. 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? Erklären 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 9.2. Schreiben Sie eine Klasse `Stoppuhr` welche zur Simulation einer Stoppuhr dienen soll. Die Stoppuhr bestehe dabei aus zwei Knöpfen. Wird der erste Knopf gedrückt, so soll die Zeitmessung gestartet werden. Wird dieser Knopf nochmals gedrückt, wird die Zeitmessung gestoppt. Der zweite Knopf dient dazu die Zeit wieder zurückzusetzen. Schreiben Sie dazu die Methoden `pushButtonStartStop` und `pushButtonReset`. Implementieren Sie weiters eine Methode, welche die verstrichene Zeit im Format `hh:mm:ss.xx` ausgibt (Beträgt die gemessene Zeit also zwei Minuten so soll `00:02:00.00` ausgegeben werden). Sie können diese Stoppuhr nun dazu verwenden Zeitmessungen durchzuführen. Speichern Sie den Source-Code unter `stoppuhr.{hpp,cpp}` in das Verzeichnis `serie09`.

Hinweis: Verwenden Sie den Datentyp `clock_t` und die Funktion `clock()` aus der Bibliothek `time.h`. Vermutlich ist es auch sinnvoll, eine Variable `isRunning` vom Typ `bool` einzuführen. Bei Betätigen des ersten Knopfes wird diese Variable entweder von `false` auf `true` gesetzt oder umgekehrt.

Aufgabe 9.3. Eine obere Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ mit

$$U = \begin{pmatrix} u_{00} & u_{01} & u_{02} & \cdots & u_{0,n-1} \\ & u_{11} & u_{12} & \cdots & u_{1,n-1} \\ & & u_{22} & \cdots & u_{2,n-1} \\ & & & \ddots & \vdots \\ \mathbf{0} & & & & u_{n-1,n-1} \end{pmatrix}$$

hat höchstens $\frac{n(n+1)}{2} = \sum_{j=1}^n j$ nicht-triviale Einträge. Schreiben Sie eine Klasse `matrixU`, in der neben der Dimension $n \in \mathbb{N}$ die Koeffizienten U_{ij} in einem dynamischen Vektor `coeff` der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Speichern Sie U spaltenweise, d.h. $U_{ij} = \text{coeff}[j*(j+1)/2+i]$ für $i, j = 0, \dots, n-1$. Wie kommt man auf diese Formel? Implementieren Sie die folgenden Funktionalitäten: Konstruktor, Destruktor, `get` und `set`-Methoden für die Matrix-Einträge, `get`-Methode für die Dimension. Schreiben Sie auch ein `main`-Programm, in welchem Sie die Implementierung testen. Speichern Sie den Source-Code unter `matrixU.{hpp/cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.4. Schreiben Sie eine Funktion um das Matrix-Vektor-Produkt $b = Ux \in \mathbb{R}^n$ für eine obere Dreiecksmatrix U und einen Vektor x bei passenden Dimensionen zu berechnen, d.h.

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

Verwenden Sie die Klasse `MatrixU` aus Aufgabe 9.3 und die Klasse `Vector` aus der Vorlesung. Stellen Sie mittels `assert` sicher, dass die Dimensionen passen. Beachten Sie, dass die Funktion nur auf Koeffizienten U_{jk} mit $0 \leq j \leq k \leq n-1$ zugreifen kann. Schreiben Sie auch ein `main`-Programm, in welchem Sie die Implementierungen testen. Speichern Sie den Source-Code unter `multMatrixVectorU.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.5. 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 oberer Dreiecksmatrizen $A, B \in \mathbb{R}^{n \times n}$ eine obere Dreiecksmatrix ist. Schreiben Sie eine Funktion, um das Matrixprodukt für zwei obere Dreiecksmatrizen bei passenden Dimensionen zu berechnen. Verwenden Sie die Klasse `MatrixU` aus Aufgabe 9.3. Stellen Sie mittels `assert` sicher, dass die Matrizen dieselbe Dimension haben. Beachten Sie, dass die Funktion nur Koeffizienten C_{jk} für $0 \leq j \leq k \leq n-1$ 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 `multMatrixMatrixU.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.6. Gegeben sei eine obere Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ mit $U_{jj} \neq 0$ für alle $j = 0, \dots, n-1$. Zu gegebenem $b \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Ux = b$. Leiten Sie eine Formel her, um die Lösung $x \in \mathbb{R}^n$ von $Ux = b$ zu berechnen, indem Sie die Formel des Matrix-Vektor-Produkts mithilfe der Dreiecksstruktur von U vereinfachen. Implementieren Sie eine Funktion, um für eine obere Dreiecksmatrix $U \in \mathbb{R}^{n \times n}$ und einen Vektor $b \in \mathbb{R}^n$ das System $Ux = b$ zu lösen. U ist dabei vom Typ `MatrixU` aus Aufgabe 9.3 und b ist dabei vom bekannten Typ `Vector` aus der Vorlesung. Stellen Sie mittels `assert` sicher, dass die Dimensionen passen und dass $U_{jj} \neq 0$ für alle j gilt. Schreiben Sie auch ein `main`-Programm, in welchem Sie die Implementierung testen. Speichern Sie den Source-Code unter `solveMatrixU.cpp` in das Verzeichnis `serie09`.

Aufgabe 9.7. Schreiben Sie eine Klasse `Matrix` zur Speicherung von Matrizen der Größe $m \times n$. Die Einträge sollen hierbei als langer `double*`-Vektor der Länge mn gespeichert werden (spaltenweise). Schreiben Sie Methoden, welche die Möglichkeit bieten Einträge zu erstellen bzw. auszulesen. Schreiben Sie einen Konstruktor, der bei übergebenen Werten $m, n \in \mathbb{N}$ eine Nullmatrix der Größe $m \times n$ erzeugt, sowie einen

Standardkonstruktor, der eine leere Matrix der Größe 0×0 generiert. Natürlich darf auch ein entsprechender Destruktor nicht fehlen. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen. Speichern Sie den Source-Code unter `matrix.{hpp/cpp}` in das Verzeichnis `serie09`.

Aufgabe 9.8. Erweitern Sie die Klasse `Matrix` aus Aufgabe 9.7 mit

- einer Methode `scanMatrix(int n,int m)`, die eine $n \times m$ -Matrix von der Tastatur einliest,
- einer Methode `printMatrix()`, die die Matrix am Bildschirm ausgibt,
- einer Methode `int typeMatrix()` so definiert: `typeMatrix()=-1` für $m \neq n$, `typeMatrix()=1` für $m = n$ und eine obere Dreiecksmatrix, `typeMatrix()=2` für $m = n$ und eine untere Dreiecksmatrix, `typeMatrix()=3` für $m = n$ und eine symmetrische Matrix, `typeMatrix()=0` sonst,
- und `columnsumnorm`, die die Spaltensummennorm

$$\|A\| = \max_{k=1,\dots,n} \sum_{j=1}^m |A_{jk}|$$

zurückgibt.

Schreiben Sie weiters ein aufrufendes main-Programm um ihre implementierten Methoden zu testen.