Dirk Praetorius
Michele Ruggeri
Bernhard Stiftner

Sommersemester 2016
07.06.2016

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 12

Aufgabe 12.1. Eine untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit

$$L = \begin{pmatrix} \ell_{11} & & & & \mathbf{0} \\ \ell_{21} & \ell_{22} & & & \\ \ell_{31} & \ell_{32} & \ell_{33} & & \\ \vdots & \vdots & \vdots & \ddots & \\ \ell_{n1} & \ell_{n2} & \ell_{n3} & \dots & \ell_{nn} \end{pmatrix}$$

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 L_{ij} in einem dynamischen Vektor der Länge $\frac{n(n+1)}{2}$ gespeichert werden. Speichern Sie L zeilenweise. Implementieren Sie die folgenden Funktionalitäten:

- Konstruktor, Copy-Konstruktor, Destruktor,
- Zuweisungssoperator,
- Zugriff auf die Koeffizienten mittels `L(i,j)` und
- die Möglichkeit eine untere Dreiecksmatrix `L` mit `cout << L` auszugeben.

Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen.

Aufgabe 12.2. Überladen Sie den Operator `+` für die Klasse `MatrixL` aus Aufgabe 12.1 um zwei untere Dreiecksmatrizen bei passenden Dimensionen addieren zu können. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen.

Aufgabe 12.3. Beweisen Sie mit der Formel des Matrix-Matrix-Produktes, dass das Produkt zweier unterer Dreiecksmatrizen eine untere Dreiecksmatrix ist. Überladen Sie den Operator `*` für die Klasse `MatrixL` aus Aufgabe 12.1 sodass Sie das Matrixprodukt für zwei untere Dreiecksmatrizen bei passenden Dimensionen berechnen können. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierungen testen.

Aufgabe 12.4. Gegeben sei eine untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ mit $\ell_{jj} \neq 0$ für alle $j = 1, \dots, n$. Zu gegebenem $b \in \mathbb{R}^n$ existiert dann ein eindeutiges $x \in \mathbb{R}^n$ mit $Lx = b$. Implementieren Sie die Möglichkeit, für eine untere Dreiecksmatrix $L \in \mathbb{R}^{n\times n}$ und einen Vektor $b \in \mathbb{R}^n$ das System $Lx = b$ mittels `x=L|b` zu lösen. L ist dabei vom Typ `Matrix` aus Aufgabe 12.1 und b ist dabei vom bekannten Typ `Vector` aus der Vorlesung. Schreiben Sie auch ein main-Programm, in welchem Sie die Implementierung testen.

Aufgabe 12.5. Welchen Aufwand hat das Lösen eines linearen Gleichungssystems aus Aufgabe 12.4? Schreiben Sie das Ergebnis in der $\mathcal{O}$-Notation auf und erklären Sie wie sie auf das Ergebnis gekommen sind.

Aufgabe 12.6. Implementieren Sie eine Klasse `Person`, welche die Datenfelder `name` und `adresse` enthält. Leiten Sie von dieser Klasse eine Klasse `Student` ab, welche die zusätzlichen Datenfelder `matrikelnummer` und `studium` enthält. Leiten Sie von der Klasse `Person` auch eine Klasse `Arbeiter` ab. Erweitern Sie diese Klasse um die Datenfelder `gehalt` und `arbeit`. Schreiben Sie für alle Klassen die zugehörigen Zugriffsfunktionen, Konstruktoren und Destruktoren. Schreiben Sie ein main-Programm und testen Sie ihre Implemtierung!

Aufgabe 12.7. Erstellen Sie für die Basisklasse `Person` aus Aufgabe 12.6 eine Methode `void print()`, welche den Namen und die Adresse einer Person am Bildschirm ausgibt. Redefinieren Sie diese Funktion dann jeweils für die Klassen `Student` und `Arbeiter` (Es sollen die zusätzlich definierten Datenelemente auch ausgegeben werden). Schreiben Sie dann noch ein Hauptprogramm, in welchem die `print`-Funktionen der verschiedenen Klassen getestet werden sollen.

Aufgabe 12.8. Wie lautet die Ausgabe des folgenden Programms? Erklären Sie warum!

```
#include <iostream>
using std::cout;
using std::endl;
class Basisklasse {
    protected:
        int N;
    public:
        Basisklasse() {
            N = 0;
            cout << "Standardkonstr. Basisklasse" << endl;
        }
        Basisklasse( int n) {
            N = n;
            cout << "Konstr. Basisklasse, N = " << N << endl;
        }
        ~Basisklasse(){
            cout << "Destr. Basisklasse, N = " << N << endl;
        }
        Basisklasse( const Basisklasse& rhs) {
            N = rhs.N;
            cout << "Kopierkonstr. Basisklasse" << endl;
        }
        Basisklasse& operator=(const Basisklasse& rhs) {
            N = rhs.N;
            cout << "Zuweisungsoperator Basisklasse" << endl;
            return *this;
        }
        int getN() const { return N; }
        void setN( int N ) { this->N = N; }
};

class Abgeleitet : public Basisklasse {
    public:
        Abgeleitet(){
            cout << "Standardkonstr. Abgeleitet" << endl;
        }
        Abgeleitet( int n):Basisklasse(n) {
            cout << "Konstr. Abgleitet, N = " << N << endl;
        }
        ~Abgeleitet() {
            cout << "Destr. Abgeleitet, N = " << N << endl;
        }
        Abgeleitet( const Abgeleitet& rhs) {
            N = rhs.N+7;
            cout << "Kopierkonstr. Abgeleitet" << endl;
        }
        Abgeleitet& operator=(const Abgeleitet& rhs) {
            N = rhs.N;
            cout << "Zuweisungsoperator Abgeleitet" << endl;
```

```
            return *this;
        }
};


Abgeleitet foo(Abgeleitet X){
    Abgeleitet tmp(5);
    tmp.setN(X.getN()*X.getN());
    return tmp;
}


int main() {
    Abgeleitet ah(10);
    {
        Abgeleitet gg(13);
        Basisklasse bs;
        Basisklasse mr=bs;
        ah=gg;
    }
    ah=foo(ah);

    return 0;
}
```