Dirk Praetorius,
Michele Ruggeri

Wintersemester 2016/17
12.01.2017

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 11

Aufgabe 11.1. 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 11.2. Erstellen Sie für die Basisklasse `Person` aus Aufgabe 11.1 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 11.3. Leiten Sie von der Klasse `Matrix` aus der Vorlesung, die Klasse `SquareMatrix` ab, welche zur Speicherung quadatrischer Matrizen dient. Diese soll die komplette Funktionalität der Klasse `Matrix` beinhalten. Testen Sie Ihren Code entsprechend!

Aufgabe 11.4. Wir betrachten die Klasse `Matrix` und die davon abgeleitete Klasse `SquareMatrix`. Implementieren Sie für die Klasse `SquareMatrix` die Methode `computeLU`, welche die LU-Zerlegung berechnet. Der Rückgabewert (Matrix $R \in \mathbb{R}^{n\times n}$) sei dabei wieder vom Type `SquareMatrix`, wobei die beiden Dreiecksmatrizen L und U in R gespeichert werden sollen. Die Diagonale von L muss hierbei nicht explizit gespeichert werden (Warum?). Nicht jede Matrix $A \in \mathbb{R}^{n\times n}$ hat eine normalisierte LU-Zerlegung $A = LU$, d.h.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ \ell_{21} & 1 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ \ell_{n1} & \dots & \ell_{n,n-1} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & \dots & u_{1n} \\ 0 & u_{22} & \ddots & \vdots \\ \vdots & \ddots & \ddots & u_{n-1,n} \\ 0 & \dots & 0 & u_{nn} \end{pmatrix}.$$

Wenn aber A eine normalisierte LU-Zerlegung besitzt, so gilt

$$u_{ik} = a_{ik} - \sum_{j=1}^{i-1} \ell_{ij} u_{jk} \quad \text{für } i = 1, \dots, n, \quad k = i, \dots, n,$$

$$\ell_{ki} = \frac{1}{u_{ii}} \Big(a_{ki} - \sum_{j=1}^{i-1} \ell_{kj} u_{ji} \Big) \quad \text{für } i = 1, \dots, n, \quad k = i+1, \dots, n,$$

$$\ell_{ii} = 1 \quad \text{für } i = 1, \dots, n,$$

wie man leicht über die Formel für die Matrix-Matrix-Multiplikation zeigen kann. Alle übrigen Einträge von $L, U \in \mathbb{R}^{n\times n}$ sind Null. Die Determinante einer Matrix $A \in R^{n\times n}$ kann über die normalisierte LU-Zerlegung berechnet werden. Es gilt namlich $\det(A) = \det(L)\det(U) = \det(U) = \prod_{j=1}^{n} u_{jj}$. Erweitern Sie die Klasse `SquareMatrix` um eine Methode `det`, die die Determinante über die LU-Zerlegung berechnet und zurückgibt. Die Matrix selbst soll hierbei nicht überschrieben werden. Testen Sie Ihren Code entsprechend!

Aufgabe 11.5. Erweitern Sie die Klasse `SquareMatrix` um eine Methode `solve`, welche die Lösung eines Gleichungssystems der Form $Ax = b$ mit Hilfe der LU-Zerlegung folgendermaßen berechnet. Für

die Matrix $A = LU$ löst man zuerst $Ly = b$ und anschließend $Ux = y$. Gleichungssysteme mit Dreiecksmatrizen können analog zu Aufgabe 9.8 gelöst werden. Testen Sie Ihren Code an einem geeigneten Beispiel.

Aufgabe 11.6. Leiten Sie von der Klasse `SquareMatrix` die Klasse `DiagonalMatrix` ab. Speichern Sie nur die Einträge auf der Diagonale. Implementieren Sie Konstruktoren, Type-Cast und den Koeffizientenzugriff. Für jene Einträge A_{ij} mit $i \neq j$ gehen Sie folgendermaßen vor: Speichern Sie zusätzliche private members `double zero` und `double const_zero`, die Sie beim entsprechenden Koeffizientenzugriff dann zurückgeben. Das heißt, bei Aufruf des `()`-Operators für `const`-Objekte wird `const_zero`, und bei normalem Aufruf wird `zero` zurückgegeben. Achten Sie, `zero` bei jedem nicht-`const` Aufruf des `()`-Operators wieder auf 0 zu setzen. Warum? Testen Sie Ihren Code entsprechend!

Aufgabe 11.7. Redefinieren Sie die Methoden `det` und `solve` aus Aufgabe 11.4 bzw. Aufgabe 11.5 sodass Sie die Determinante von Diagonalmatrizen berechnen können bzw. lineare Gleichungssysteme $Ax = b$ lösen können. Nützen Sie dabei die diagonale Struktur der Matrix aus. Welchen Aufwand hat das Lösen dann? Schreiben Sie das Ergebnis in der $\mathcal{O}$-Notation. Erklären Sie, wie Sie auf das Ergebnis gekommen sind! Testen Sie Ihren Code entsprechend!

Aufgabe 11.8. 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;
}
```