Dirk Praetorius,
Gregor Gantner

Sommersemester 2018
19.06.2018

Übungen zur Vorlesung
Einführung in das Programmieren für TM

Serie 12

Aufgabe 12.1. 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 Implementierung!

Aufgabe 12.2. Erstellen Sie für die Basisklasse `Person` aus Aufgabe 12.1 eine Methode `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 ein main-Programm, in welchem die `print`-Funktionen der verschiedenen Klassen getestet werden sollen.

Aufgabe 12.3. 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 12.4. Wir betrachten die Klasse `Matrix` und die davon abgeleiteten Klassen `Vector` aus der Vorlesung und `SquareMatrix` aus Aufgabe 12.3. Implementieren Sie für die Klasse `SquareMatrix` die Methode `solve`, welche das lineare Gleichungssystem $Ax = b$ mit dem *Gauß'schen Eliminationsverfahren* löst. Gegeben seien eine Matrix $A \in \mathbb{R}^{n\times n}$ (Typ `SquareMatrix`) und eine rechte Seite $b \in \mathbb{R}^n$ (Typ `Vector`):

- Zunächst bringt man die Matrix A auf obere Dreiecksform, indem man die Unbekannten eliminiert. Gleichzeitig modifiziert man die rechte Seite b.
- Das entstandene Gleichungssystem mit oberer Dreiecksmatrix A löst man direkt.

Im ersten Eliminationsschritt zieht man geeignete Vielfache der ersten Zeile von den übrigen Zeilen ab und erhält dadurch eine Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ 0 & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ 0 & a_{n2} & \dots & a_{nn} \end{pmatrix}.$$

Im zweiten Eliminationsschritt zieht man nun geeignete Vielfache der zweiten Zeile von den übrigen Zeilen ab und erhält eine Matrix der Form

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ 0 & a_{22} & a_{23} & \dots & a_{2n} \\ 0 & 0 & a_{33} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & a_{n3} & \dots & a_{nn} \end{pmatrix}.$$

Nach $n-1$ Eliminationsschritten erhält man also eine obere Dreiecksmatrix A. Stellen Sie sich mittels `assert` sicher, dass $a_{kk} \neq 0$ im k-ten Eliminationsschritt gilt. Berücksichtigen Sie, dass auch die rechte Seite $b \in \mathbb{R}^n$ geeignet modifiziert werden muss. Lösen Sie das System $Ax = b$ mit A oberen

Dreiecksmatrix direkt (analog zu unteren Dreiecksmatrizen wie in Aufgabe 11.5). Welchen Aufwand hat Ihre Implementierung des Gauß'schen Eliminationsverfahren und warum? Machen Sie sich das Vorgehen zunächst an einem Beispiel mit $A \in \mathbb{R}^{2\times 2}$ sowie $A \in \mathbb{R}^{3\times 3}$ klar. Testen Sie Ihren Code entsprechend!

Aufgabe 12.5. Das Gauß'sche Eliminationsverfahren aus Aufgabe 12.4 scheitert, falls im k-ten Schritt $a_{kk} = 0$ gilt, auch wenn das Gleichungssystem $Ax = b$ eine eindeutige Lösung x besitzt. Deshalb kann man das Verfahren um eine sogenannte *Pivot-Suche* erweitern:

- Im k-ten Schritt wählt man aus $a_{kk}, \ldots, a_{nk}$ das betragsgrößte Element a_{pk}.
- Dann vertauscht man die k-te und die p-te Zeile von A (und b).
- Schließlich führt man den Eliminationsschritt aus wie zuvor.

Implementieren Sie für die Klasse `SquareMatrix` aus Aufgabe 12.3 die Methode `gausspivot`, die die Lösung von $Ax = b$ wie angegeben berechnet. (Man kann übrigens mathematisch beweisen, dass das Gauss-Verfahren mit Pivot-Suche genau dann durchführbar ist, wenn das Gleichungssystem $Ax = b$ eine eindeutige Lösung besitzt. Einen Beweis dazu sehen Sie in der Vorlesung zur Numerischen Mathematik.) Testen Sie Ihren Code entsprechend!

Aufgabe 12.6. Leiten Sie von der Klasse `SquareMatrix` aus Aufgabe 12.3 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 darauf, `zero` bei jedem nicht-`const` Aufruf des `()`-Operators wieder auf 0 zu setzen. Warum? Testen Sie Ihren Code entsprechend!

Aufgabe 12.7. Redefinieren Sie die Methode `solve` aus Aufgabe 12.4 für die Klasse `DiagonalMatrix` aus Aufgabe 12.6 sodass Sie ein lineares Gleichungssystem $Ax = b$ für eine DiagonalMatrix A lösen können. Nützen Sie dabei die diagonale Struktur der Matrix A aus. Welchen Aufwand hat das Lösen und warum? Testen Sie Ihren Code entsprechend!

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. Abgeleitet, 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;
}
```