RHEINISCH-
WESTFÄLISCHE
TECHNISCHE
HOCHSCHULE
AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK 2
RWTH Aachen · D-52056 Aachen · GERMANY
http://programmierung.rwth-aachen.de/
Prof. Dr. Jürgen Giesl

Prof. Dr. Jürgen Giesl
Carsten Fuhs, Peter Schneider-Kamp, Stephan Swiderski

# Prüfungsklausur Programmierung

**26. 3. 2008**

**Vorname:** ______________________________

**Nachname:** ______________________________

**Matrikelnummer:** ______________________________

**Studiengang (bitte ankreuzen):**

○ Informatik Bachelor ○ Informatik Diplom ○ Informatik Lehramt

○ Mathematik Bachelor

○ Sonstige: ______________________________

- Schreiben Sie bitte auf jedes Blatt **Vorname**, **Name** und **Matrikelnummer**.
- Geben Sie Ihre Antworten bitte in lesbarer und verständlicher Form an. Schreiben Sie bitte nicht mit roten Stiften oder mit Bleistiften.
- Bitte beantworten Sie die Aufgaben auf den **Aufgabenblättern**. Benutzen Sie ggf. auch die Rückseiten der **zur jeweiligen Aufgabe gehörenden** Aufgabenblätter.
- Antworten auf anderen Blättern können nur berücksichtigt werden, wenn **Name, Matrikelnummer und Aufgabennummer** deutlich darauf erkennbar sind.
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch **Durchstreichen**.
- Werden Täuschungsversuche beobachtet, so wird die Klausur mit **0 Punkten** bewertet.
- Geben Sie bitte am Ende der Klausur **alle Blätter** zusammen mit den Aufgabenblättern ab.

| | Anzahl Punkte | Erreichte Punkte |
|---|---|---|
| Aufgabe 1 | 14 | |
| Aufgabe 2 | 12 | |
| Aufgabe 3 | 12 | |
| Aufgabe 4 | 28 | |
| Aufgabe 5 | 17 | |
| Aufgabe 6 | 17 | |
| Summe | 100 | |
| Prozentzahl | | |

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



**Aufgabe 1 (Programmanalyse,** 8 + 6 Punkte**)**

a) Geben Sie die Ausgabe des Programms für den Aufruf `java M` an. Schreiben Sie hierzu jeweils die ausgegebenen Zeichen hinter den Kommentar "`OUT:`".

```
public class A {
    protected int x = 1;
    public A() {
        this(4);
    }
    public A(int x) {
        this.x += x;
    }
    protected void f(B b) {
        this.x += b.x;
    }
}

public class B extends A {
    protected static int y = 3;
    public B() {
    }
    public B(int x) {
        super(x);
        y--;
    }
    protected void f(B b) {
        x += b.y;
    }
    protected void f(A a) {
        a = this;
    }
}

public class M {
    public static void main(String[] args) {
        A a = new A();
        System.out.println(a.x);          // OUT: 5
        B b = new B(10);
        System.out.println(b.x+" "+b.y); // OUT: 11 2
        a.f(b);
        System.out.println(a.x);          // OUT: 16
        b.f(b);
        System.out.println(b.x+" "+b.y); // OUT: 13 2
        b.f(a);
        System.out.println(a.x);          // OUT: 16
        b.f(new B());
        System.out.println(b.x);          // OUT: 15
    }
}
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



b) Wir schreiben zusätzlich zu `A` und `B` eine neue Klasse `C`. Welche drei Fehler treten beim Compilieren auf? Begründen Sie Ihre Antwort kurz.

```
public class C extends B {
    public C() {
        super(3);
    }
    public static void f(A a) {
        final double z;
        B b = new C();
        z = b.y;
        int x = A.x;
        b = new A();
    }
}
```

- *Die Methode* `f(A a)` *in* `B` *kann nicht durch eine statische Methode überschrieben werden.*
- *Das Attribut* `x` *ist kein statisches Attribut der Klasse* `A`*.*
- *Die Variable* `b` *hat den Typ* `B` *und kann deshalb kein Objekt der Oberklasse* `A` *aufnehmen.*

*Man beachte, dass das einmalige Setzen einer* `final`*–Variablen kein Fehler ist!*

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



**Aufgabe 2 (Verifikation,** 10 + 2 Punkte**)**

Der Algorithmus $P$ berechnet für eine ganze Zahl $n \in \mathbb{Z}$ den Wert $n^3$.

a) Vervollständigen Sie die folgende Verifikation des Algorithmus $P$ im Hoare-Kalkül, indem Sie die unterstrichenen Teile ergänzen. Hierbei dürfen zwei Zusicherungen nur dann direkt untereinander stehen, wenn die untere aus der oberen folgt. Hinter einer Programmanweisung darf nur eine Zusicherung stehen, wenn dies aus einer Regel des Hoare-Kalküls folgt.

**Algorithmus:** $P$
**Eingabe:** $n \in \mathbb{Z}$
**Ausgabe:** $res$
**Vorbedingung:** $true$
**Nachbedingung:** $res = n^3$

$\langle\ true\ \rangle$

$\langle\ n * n = n * n\ \rangle$

$p = n * n;$

$\langle\ p = n * n\ \rangle$

$\langle\ p = n * n \land 0 = 0\ \rangle$

$res = 0;$

$\langle\ p = n * n \land res = 0\ \rangle$

$\langle\ p \geq 0 \land res = (n^2 - p) * n\ \rangle$

`while` $(p > 0)$ {

$\langle\ p \geq 0 \land res = (n^2 - p) * n \land p > 0\ \rangle$

$\langle\ p - 1 \geq 0 \land res + n = (n^2 - p) * n + n\ \rangle$

$res = res + n;$

$\langle\ p - 1 \geq 0 \land res = (n^2 - p) * n + n\ \rangle$

$\langle\ p - 1 \geq 0 \land res = (n^2 - (p - 1)) * n\ \rangle$

$p = p - 1;$

$\langle\ p \geq 0 \land res = (n^2 - p) * n\ \rangle$

}

$\langle\ p \geq 0 \land res = (n^2 - p) * n \land p \not> 0\ \rangle$

$\langle\ res = n^3\ \rangle$

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



b) Beweisen Sie die Terminierung des Algorithmus $P$.

Wir wählen die Variante $p$. Dann gilt $p > 0 \Rightarrow p \geq 0$ und

$$\langle\ p = m \wedge p > 0\ \rangle$$

$$\langle\ p - 1 < m\ \rangle$$

$res = res + n;$

$$\langle\ p - 1 < m\ \rangle$$

$p = p - 1;$

$$\langle\ p < m\ \rangle$$

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |

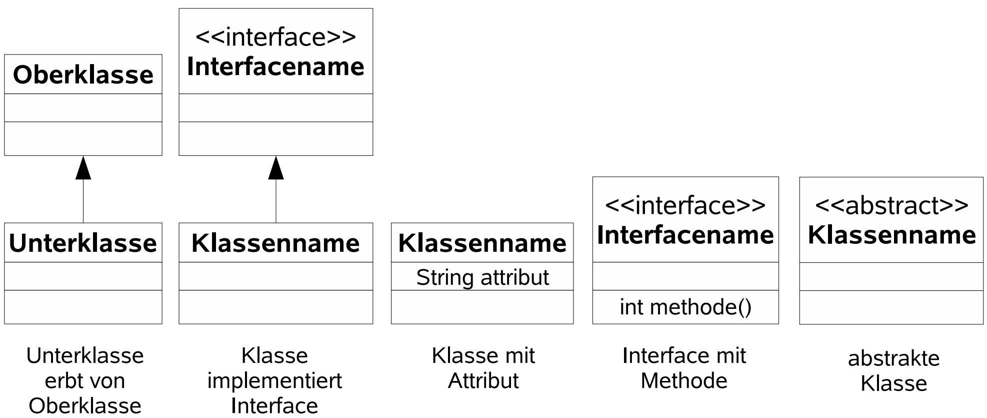


**Aufgabe 3 (Datenstrukturen in Java,** 6 + 6 Punkte)

Ihre Aufgabe ist es, eine objektorientierte Datenstruktur zur Verwaltung von Luftfahrzeugen zu entwerfen. Bei der vorangehenden Analyse wurden folgende Eigenschaften der verschiedenen Luftfahrzeuge ermittelt.

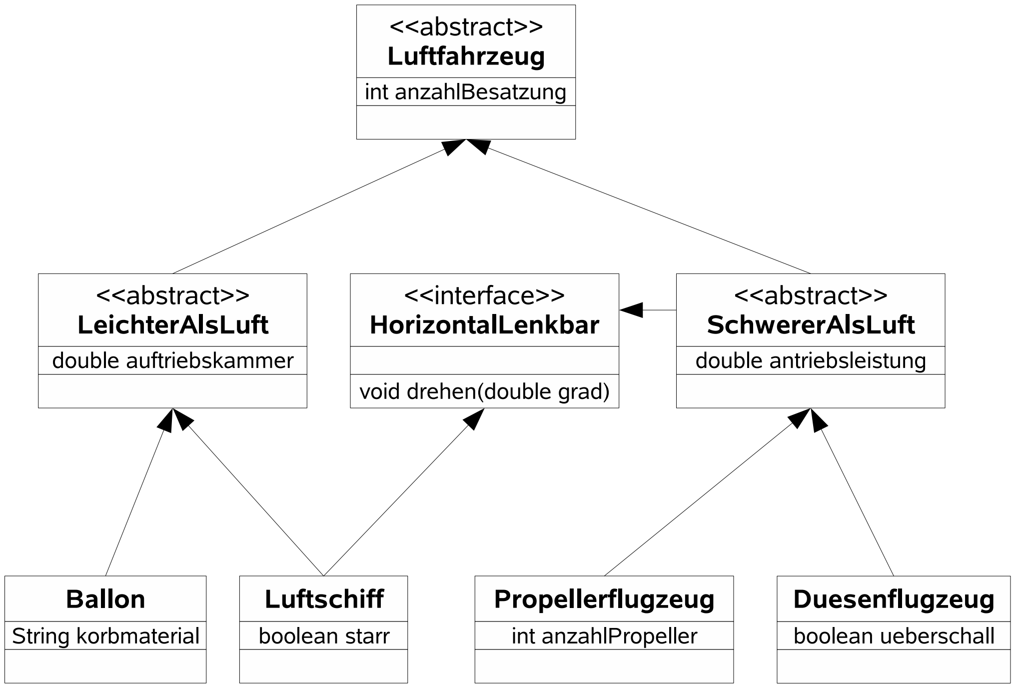
- Ein Propellerflugzeug ist ein Luftfahrzeug der Kategorie „Schwerer als Luft". Jedes Propellerflugzeug hat eine bestimmte Anzahl von Besatzungsmitgliedern. Weiterhin sind bei einem Propellerflugzeug auch die Anzahl der Propeller und die Leistung des Antriebs interessant.
- Düsenflugzeuge sind ebenfalls Luftfahrzeuge der Kategorie „Schwerer als Luft". Auch sie haben eine bestimmte Anzahl an Besatzungsmitgliedern und eine bestimmte Antriebsleistung. Bei Düsenflugzeugen ist zudem die Information wichtig, ob das Düsenflugzeug mit Überschallgeschwindigkeit fliegen kann.
- Bei Luftfahrzeugen der Kategorie „Leichter als Luft" ist das Volumen der Auftriebskammer eine wichtige Kenngröße. Außerdem ist auch bei Luftfahrzeugen der Kategorie „Leichter als Luft" die Anzahl der Besatzungsmitglieder von Interesse.
- Ein Ballon ist ein Luftfahrzeug der Kategorie „Leichter als Luft". Bei einem Ballon ist vor allem das Material interessant, aus dem sein Korb besteht.
- Ein Luftschiff ist ebenfalls ein Luftfahrzeug der Kategorie „Leichter als Luft". Es gibt sowohl Luftschiffe mit einer starren Hülle als auch Luftschiffe mit einer nicht-starren Hülle.
- Alle Luftfahrzeuge der Kategorie „Schwerer als Luft" sowie Luftschiffe sind lenkbar. Man kann ihre Bewegungsrichtung in der Horizontalen mit einer Methode ändern, der man als Parameter den Winkel übergibt, um den die Bewegungsrichtung geändert werden soll.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Luftfahrzeugen. Achten Sie darauf, dass gemeinsame Merkmale in (evtl. abstrakten) Oberklassen zusammengefasst werden. Notieren Sie Ihren Entwurf graphisch und verwenden Sie dazu die folgende Notation:




Geben Sie für jede Klasse ausschließlich den jeweiligen Namen und die Namen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen seiner Methoden an.

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



b) Implementieren Sie in Java eine Methode `voelligLosgeloest`. Die Methode bekommt als Parameter ein Array von Luftfahrzeugen übergeben. Sie soll die Anzahl der Düsenflugzeuge im Array liefern, bei denen die Anzahl an Besatzungsmitgliedern genau 1 ist und die zugleich mit Überschallgeschwindigkeit fliegen können.

Gehen Sie dabei davon aus, dass das übergebene Array nicht der `null`-Wert ist, dass es keine `null`-Werte enthält und dass für alle Attribute geeignete Selektoren existieren. Verwenden Sie für den Zugriff auf die benötigten Attribute die passenden Selektoren und kennzeichnen Sie die Methode mit dem Schlüsselwort „`static`“, falls angebracht.

```
public static int voelligLosgeloest(Luftfahrzeug[] array) {
  int ergebnis = 0;

  for (int i = 0; i < array.length; i++) {
       if (array[i] instanceof Duesenflugzeug  &&
           array[i].getAnzahlBesatzung() == 1  &&
           ((Duesenflugzeug) array[i]).getUeberschall())
       ergebnis++;
  }

  return ergebnis;
}
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



**Aufgabe 4 (Rekursive Datenstrukturen,** 8 + 8 + 12 Punkte)

Die Klasse `Liste` dient zur Repräsentation von Listen von Zahlen. Jedes Listenelement wird als Objekt der Klasse `Element` dargestellt. Ein Listenelement enthält eine Zahl und einen Verweis auf den Nachfolger. Die Zahl wird in dem Attribut `wert` gespeichert und das Attribut `nachfolger` zeigt auf das nächste Element der Liste. Das letzte Element einer Liste hat keinen Nachfolger, so dass dessen Attribut `nachfolger` auf `null` zeigt. Objekte der Klasse `Liste` haben ein Attribut `start`, das auf das erste Element der Liste zeigt. Eine leere Liste hat kein erstes Element, so dass hier das Attribut `start` auf `null` zeigt.

```
public class Liste {
    Element start;

    public Liste(Element start) {
        this.start = start;
    }

}

public class Element {
    int wert;
    Element nachfolger;

    public Element(int wert, Element nachfolger) {
        this.wert = wert;
        this.nachfolger = nachfolger;
    }
}
```

| Vorname | Name | Matr.-Nr. |
| --- | --- | --- |
| | | |



a) Implementieren Sie die Methode

```
public Liste umdrehen()
```

der Klasse `Liste`. Die Methode `umdrehen` soll eine *neue* Liste zurückliefern, welche alle Werte der aktuellen Liste in umgekehrter Reihenfolge enthält. Falls die aktuelle Liste `xs` die Liste $[\mathtt{x}_1, \mathtt{x}_2, \ldots, \mathtt{x}_\mathtt{n}]$ ist, sollte `xs.umdrehen()` demnach die *neue* Liste $[\mathtt{x}_\mathtt{n}, \ldots, \mathtt{x}_2, \mathtt{x}_1]$ zurückliefern. Diese neue Liste Liste sollte also insbesondere andere `Element`-Objekte als die Ursprungsliste `xs` enthalten. Die Liste `xs` und ihre `Element`-Objekte sollten nicht verändert werden.

Verwenden Sie bei der Implementierung ***keine*** *Rekursion*, sondern ***ausschließlich*** *Schleifen.*

```
public Liste umdrehen(){
    Element aktuelles = this.start;
    Element element = null;
    while (aktuelles != null){
        element = new Element(aktuelles.wert,element);
        aktuelles = aktuelles.nachfolger;
    }
    return new Liste(element);
}
```

| Vorname | Name | Matr.-Nr. |
| --- | --- | --- |
| | | |



b) Das Interface `Filter` hat eine Methode `filtereZahl`, die für eine Zahl `true` oder `false` zurückliefert.

```
public interface Filter {

    public boolean filtereZahl(int wert);
}
```

Implementieren Sie die Methode

```
    public Liste filtereListe(Filter f)
```

der Klasse `Liste`. Sie soll aus der aktuellen Liste eine *neue* Liste berechnen. Die *neue* Liste enthält nur die Werte, für die der Filter `true` liefert. Falls die aktuelle Liste `xs` also $[x_1, \ldots, x_n]$ ist, so sollte `xs.filtereListe(f)` eine neue Liste zurückliefern, die alle Werte $x_i$ enthält, für die `f.filtereZahl`($x_i$) das Ergebnis `true` liefert. Die Reihenfolge der Werte soll in der *neuen* Liste erhalten bleiben. Die Liste `xs` und ihrere Elemente sollen hierbei nicht verändert werden.

Sei beispielsweise `xs` die Liste $[6, 2, 7, 3, 1, 4]$ und `f` ein Objekt des Typs `Filter`. Sei `f.filtereZahl` die Methode, bei der `f.filtereZahl`(x) das Ergebnis `true` liefert, wenn `x` gerade ist und ansonsten `false`. Der Aufruf `xs.filtereListe(f)` sollte dann die neue Liste $[6, 2, 4]$ zurückliefern.

Verwenden Sie bei Ihrer Implementierung ***keine*** *Schleifen*, sondern ***ausschließlich*** *Rekursion*. Sie dürfen zusätzliche Hilfsmethoden einführen. Markieren Sie die Methoden wenn möglich mit `static`.

```
    public Liste filtereListe(Filter f){
        return new Liste(filtereElement(this.start,f));
    }

    private static Element filtereElement(Element elem, Filter f) {
        if (elem == null) return null;
        if (f.filtereZahl(elem.wert)){
            return new Element(elem.wert,filtereElement(elem.nachfolger, f));
        } else {
            return filtereElement(elem.nachfolger, f);
        }
    }
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



c) Implementieren Sie die Methode

```
public Liste groesserAls(int x)
```

der Klasse `Liste`. Sie soll aus der aktuellen Liste und dem Eingabewert `x` eine *neue* Liste berechnen, welche alle Werte der aktuellen Liste enthält, die größer als `x` sind. Die aktuelle Liste sowie ihre Elemente sollen dabei nicht verändert werden. Wenn beispielsweise `xs` die Liste $[6, 2, 7, 3, 1, 4]$ ist, liefert der Aufruf `xs.groesserAls(3)` die Liste $[6, 7, 4]$ zurück.

Verwenden Sie bei der Implementierung ***keine*** *Schleifen* und auch ***keine*** *Rekursion*. Schreiben Sie stattdessen eine geeignete Klasse `GroesserFilter`, die das Interface `Filter` implementiert. Die Methode `groesserAls` aus der Klasse `Liste` sollte dann die Methode `filtereListe` aus Aufgabenteil b) sowie Ihre Klasse `GroesserFilter` verwenden.

*Hinweis:* Die Klasse `GroesserFilter` kann nicht nur Methoden, sondern auch Attribute besitzen.

```
public class GroesserFilter implements Filter{
    int x;

    public boolean filtereZahl(int wert) {
        return wert > x;
    }

    public GroesserFilter(int x) {
        this.x = x;
    }
}

public Liste groesserAls(int x){
    return filtereListe(new GroesserFilter(x));
}
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



**Aufgabe 5 (Funktionale Programmierung in Haskell,** 3 + 3 + 2 + 3 + 6 Punkte)

a) Geben Sie jeweils den allgemeinsten Typ der Funktionen `f` und `g` an, die wie folgt definiert sind. Gehen Sie davon aus, dass `2` den Typ `Int` hat.

```
f x = \y -> y+2:x

f :: [Int] -> Int -> [Int]


g h = [\x -> x,\x -> h x,\x -> h (h x)]

g :: (b -> b) -> [b -> b]
```

b) Sei das folgende Programm gegeben:

```
head :: [a] -> a
head (x:xs) = x

map :: (a -> b) -> [a] -> [b]
map f []     = []
map f (x:xs) = f x:map f xs

bot :: a
bot = bot

h :: [(Int,Char)] -> Int -> [Char]
h []          n      = []
h ((x,c):xs)  n      | x == n    = c : h xs n
                     | otherwise = h xs n
```

Bestimmen Sie das Ergebnis der Auswertung für die beiden folgenden Ausdrücke.

```
head (map (\x->x+1) [3*2,2,bot])
```

7

```
h [(0,'y'),(3,'G'),(5,'r'),(2,'u'),(1,'r'),(3,'O')] 3
```

"GO" oder ['G',O']

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



c) Ein boolescher Ausdruck besteht aus den binären Operatoren $\vee$ und $\wedge$ sowie den Konstanten $W$ und $F$ (für "wahr" und "falsch"). Formal ist die Menge $\mathcal{B}$ der booleschen Ausdrücke induktiv wie folgt definiert:

- $W, F \in \mathcal{B}$
- $(s \vee t) \in \mathcal{B}$, wenn $s, t \in \mathcal{B}$
- $(s \wedge t) \in \mathcal{B}$, wenn $s, t \in \mathcal{B}$

Gültige Boolesche Ausdrücke sind beispielsweise folgende:

- $((W \vee F) \wedge W)$
- $(F \wedge W)$
- $(F \vee W)$
- $W$

Entwerfen Sie eine geeignete Datenstruktur für boolesche Ausdrücke in Haskell.

```
data BA = W | F | Und BA BA | Oder BA BA
```

d) Implementieren Sie eine Funktion `ausrechnen` in Haskell, die den Wahrheitswert eines booleschen Ausdrucks aus $\mathcal{B}$ ausrechnet. Das Ergebnis soll vom vordefinierten Typ `Bool` sein. Dabei sollen die Operatoren $\vee$ und $\wedge$ als logisches Oder und logisches Und interpretiert werden. Die Konstanten $W$ und $F$ sollen als `True` und `False` interpretiert werden. Für den booleschen Ausdruck $((W \vee F) \wedge W)$ liefert die Funktion `ausrechnen` also das Ergebnis `True`. Geben Sie außerdem den Typ der Funktion `ausrechnen` an.

```
ausrechnen :: BA -> Bool
ausrechnen W = True
ausrechnen F = False
ausrechnen (Und x y)  = ausrechnen x && ausrechnen y
ausrechnen (Oder x y) = ausrechnen x || ausrechnen y
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



e) Implementieren Sie eine Funktion `subset :: [Int] -> [Int] -> Bool` in Haskell. Hierbei soll `subset xs ys` genau dann das Ergebnis `True` liefern, wenn alle Elemente der Liste `xs` auch in `ys` auftreten. Tritt ein Element mehrfach in `xs` auf, so muss es auch in `ys` mindestens genauso oft auftreten. Die Reihenfolge der Elemente spielt hierbei keine Rolle. Beispielsweise liefert `subset [1,2,2] [2,3,2,1]` das Ergebnis `True` und `subset [1,2,2,3] [1,2,3]` liefert `False`. Sie dürfen bei Ihrer Implementierung keine vordefinierten Funktionen außer `[]`, `:`, `==`, `&&` und `||` verwenden.

```haskell
subset :: [Int] -> [Int] -> Bool
subset [] _ = True
subset (x:xs) ys = member x ys && subset xs (delete x ys)

    where delete :: Int -> [Int] -> [Int]
          delete _ [] = []
          delete x (y:ys) | x == y = ys
                          | otherwise = y : delete x ys

          member :: Int -> [Int] -> Bool
          member x [] = False
          member x (y:ys) = x == y || member x ys
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



**Aufgabe 6 (Logische Programmierung in Prolog,** (2 + 2 + 2) + 4 + (6 + 1) Punkte**)**

a) Eine Funktion auf einem endlichen Definitionsbereich kann durch eine Liste von Paaren dargestellt werden. Wir betrachten als Beispiel die Funktion $f : \{0, 1, 2\} \rightarrow \{0, 2, 4\}$, die jede natürliche Zahl zwischen 0 und 2 verdoppelt (d.h. $f(0) = 0$, $f(1) = 2$, $f(2) = 4$). Als Liste in Prolog können wir diese Funktion beispielsweise durch die Liste `[f(0,0), f(1,2), f(2,4)]` darstellen.

- Implementieren Sie in Prolog ein dreistelliges Prädikat `lookup` zum Anwenden einer durch eine Liste von Paaren dargestellten Funktion $f$ auf einen Wert. Falls $\mathtt{y}, \mathtt{x_1}, \ldots, \mathtt{x_n}$, $\mathtt{v_1}, \ldots, \mathtt{v_n}$ Werte sind, so sollte die Anfrage "`?- lookup`$(\mathtt{y}, [\mathtt{f}(\mathtt{x_1}, \mathtt{v_1}), \ldots, \mathtt{f}(\mathtt{x_n}, \mathtt{v_n})], \mathtt{V})$." die Variable `V` an den Wert $\mathtt{v_i}$ binden, bei dem $\mathtt{y} = \mathtt{x_i}$ gilt. Falls `y` nicht im Definitionsbereich enthalten ist (d.h. falls $\mathtt{y} \neq \mathtt{x_i}$ für alle `i` gilt), so sollte die Anfrage fehlschlagen. Sie dürfen dabei davon ausgehen, dass die Liste für jedes Element des Definitionsbereichs nur ein Paar enthält (d.h. die Werte $\mathtt{x_1}, \ldots, \mathtt{x_n}$ sind alle paarweise verschieden). Beispielsweise ergibt "`?- lookup`$(1, [\mathtt{f}(0, 0), \mathtt{f}(1, 2), \mathtt{f}(2, 4)], \mathtt{V})$." die Lösung `V = 2`.

```
lookup(K,[f(K,V)|_],V).
lookup(K,[_|KVs],V) :- lookup(K,KVs,V).
```

- Implementieren Sie in Prolog ein dreistelliges Prädikat `map`. Das erste Argument von `map` ist wieder eine durch eine Liste von Paaren dargestellte Funktion $f$, d.h. eine Liste der Form $[\mathtt{f}(\mathtt{x_1}, \mathtt{v_1}), \ldots, \mathtt{f}(\mathtt{x_n}, \mathtt{v_n})]$. Das zweite Argument von `map` ist eine Liste von Werten $[\mathtt{y_1}, \ldots, \mathtt{y_m}]$, auf die die Funktion $f$ jeweils angewendet werden soll. Wenn diese beiden Argumente vorgegeben werden, so soll im dritten Argument von `map` die Liste der Ergebnisse berechnet werden, die sich ergeben, wenn man $f$ jeweils auf die Argumente $\mathtt{y_1}, \ldots, \mathtt{y_m}$ anwendet. Beispielsweise ergibt "`?- map`$([\mathtt{f}(0, 0), \mathtt{f}(1, 2), \mathtt{f}(2, 4)], [1, 2], \mathtt{Vs})$." die Lösung `Vs = [2,4]`.

```
map(_,[],[]).
map(KVs,[X|XS],[Y|YS]) :- lookup(X,KVs,Y), map(KVs,XS,YS).
```

- Geben Sie eine passende Liste `xs` an, so dass bei der Anfrage "`?- map`$(\mathtt{xs}, [1, 2, 3], \mathtt{Vs})$." die Variable `Vs` an die Liste `[1,4,9]` gebunden wird.

  *Als `xs` kann man die Liste `[f(1,1),f(2,4),f(3,9)]` wählen.*

| Vorname | Name | Matr.-Nr. |
| --- | --- | --- |
| | | |



b) Geben Sie für die folgenden Paare von Termen den allgemeinsten Unifikator an, oder begründen Sie kurz, warum dieser nicht existiert.

- `f(H,U,R,R,Y)` und `f(a,b,g(U),g(Y),b)`

```
H = a
U = b
R = g(b)
Y = b
```

- `f(A,g(B,B),g(C,C))` und `f(g(D,D),D,B)`

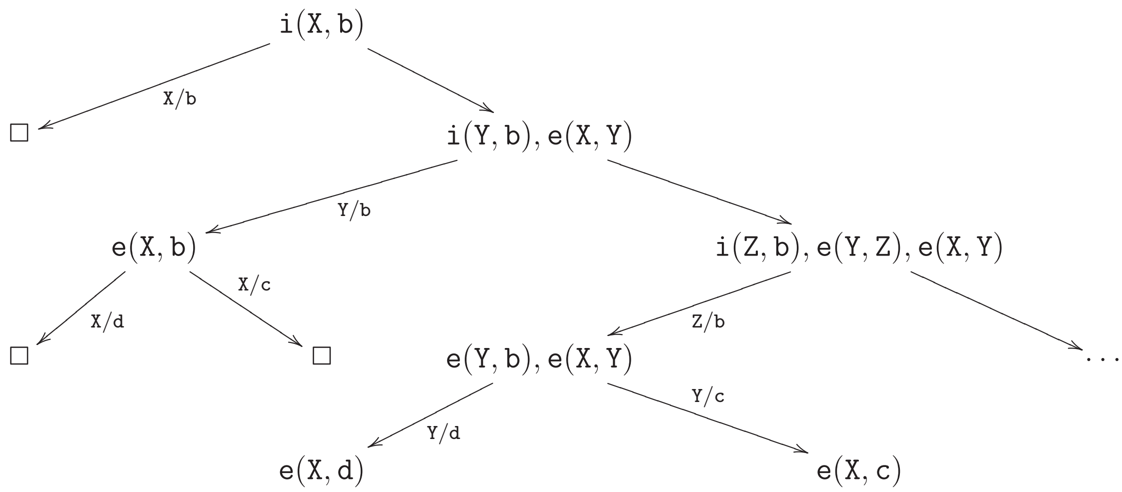
```
A = g(g(g(C,C),g(C,C)),g(g(C,C),g(C,C)))
D = g(g(C,C),g(C,C))
B = g(C,C)
```

| Vorname | Name | Matr.-Nr. |
|---|---|---|
| | | |



c) Erstellen Sie für das folgende Logikprogramm den SLD-Baum zur Anfrage "`?- i(X,b).`" und geben Sie alle Lösungen an. Sie dürfen dabei abbrechen, sobald die Anfrage aus mehr als drei Teilen (Atomen) bestehen würde. Kennzeichnen Sie solche Knoten durch "...".

```
e(b,a).
e(d,b).
e(c,b).
i(X,X).
i(X,Z) :- i(Y,Z), e(X,Y).
```




Die Lösungen sind `X=b`, `X=d` und `X=c`.

Ordnen Sie die Klauseln und Atome des Programms so an, dass Prolog bei der Anfrage "`?- i(X,b).`" terminiert, d.h., dass der SLD-Baum endlich wird.

```
e(b,a).
e(d,b).
e(c,b).
i(X,X).
i(X,Z) :- e(X,Y), i(Y,Z).
```