RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN

LEHR- UND FORSCHUNGSGEBIET INFORMATIK II
RWTH Aachen · D-52056 Aachen · GERMANY
http://www-i2.informatik.rwth-aachen.de/lufgi2
Prof. Dr. Jürgen Giesl

# Diplom-Vorprüfung Lösungsvorschlag
# Informatik I - Programmierung
**23. 9. 2002**

**Vorname:** _____

**Nachname:** _____

**Matrikelnummer:** _____

**Studiengang (bitte ankreuzen):**

○ Informatik Diplom ○ Informatik Lehramt ○ 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.
- Bitte beantworten Sie die Aufgaben auf den **Aufgabenblättern**. Benutzen Sie 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 ist.
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch **Durchstreichen**.
- Werden Täuschungsversuche beobachtet, so wird die Klausur mit **nicht bestanden** bewertet.
- Geben Sie bitte am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.

| | Anzahl Punkte | Erreichte Punkte |
|---|---|---|
| Aufgabe 1 | 18 | |
| Aufgabe 2 | 11 | |
| Aufgabe 3 | 12 | |
| Aufgabe 4 | 19 | |
| Aufgabe 5 | 16 | |
| Aufgabe 6 | 13 | |
| Summe | 89 | |
| Note | - | |

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



## Aufgabe 1 (Programmanalyse, 12 + 6 Punkte)

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

```
public class A {
  public int wert = 3;

  public static int eq(A x1, A x2) {
    if (x1 == x2) return 1; else return 2;}
  public int eq(A x) {
   if (this == x) return 3; else return 4;}
}

public class B extends A {
  public B(int wert) {
    this.wert = this.wert + wert;}
  public int eq(A x) {
    if (wert == x.wert) return 5; else return 6;}
  public int eq(B y) {
    if (this == y) return 7; else return 8;}
}

public class H {
  public static void main(String[] args) {
    A a1 = new B(5);
    System.out.println(a1.wert);        // AUSGABE: 8
    B b = (B) a1;
    A a2 = new A();
    System.out.println(a2.wert);        // AUSGABE: 3
    a2.wert = a1.wert;
    System.out.println(B.eq(a1,a2));    // AUSGABE: 2
    System.out.println(a2.eq(a1));      // AUSGABE: 4
    System.out.println(a1.eq(a2));      // AUSGABE: 5
    System.out.println(b.eq(b));}       // AUSGABE: 7
}
```

(b) Das Programm wird um eine zusätzliche Klasse `C` ergänzt. Finden und erklären Sie die drei Fehler in dieser Klasse, die das Compilieren verhindern.

```
public class C extends A {
  private static int eq(A x1, A x2) {
    return 1;}

  public double eq (A obj) {
    A x = obj;
    C c = x;
    return c.wert;}
}
```

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



- `private static int eq(A x1, A x2) ...` geht nicht, da diese `public`-Methode nicht durch eine Methode mit geringerer Sichtbarkeit überschrieben werden darf. Man müsste `private` durch `public` ersetzen.
- `public double eq (A obj) ...` geht nicht, denn beim Überladen darf nicht bloß der Rückgabetyp verschieden sein. Der Typ `double` müsste durch `int` ersetzt werden.
- `C c = x;` geht nicht, da explizit konvertiert werden muss (d.h., man müsste es z.B. durch `C c = (C) x;` ersetzen).

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



## Aufgabe 2 (Verifikation, 10 + 1 Punkte)

Der Algorithmus $P$ berechnet die Potenz $\mathtt{x}^{\mathtt{y}}$ zweier natürlicher Zahlen $\mathtt{x}$ und $\mathtt{y}$ mit $\mathtt{x} > 0$ und $\mathtt{y} > 0$.

(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:** $\mathtt{x}, \mathtt{y} \in \{1, 2, ...\}$
**Ausgabe:** $\mathtt{res}$
**Vorbedingung:** $\mathtt{x} > 0 \wedge \mathtt{y} > 0$
**Nachbedingung:** $\mathtt{res} = \mathtt{x}^{\mathtt{y}}$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{y} = \mathtt{y}\ \rangle$

```
z = y;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y}\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y} \wedge 1 = 1\ \rangle$

```
res = 1;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = \mathtt{y} \wedge \mathtt{res} = 1\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

```
while (z != 0) {
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} \neq 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} * \mathtt{x} = \mathtt{x}^{\mathtt{y}-(\mathtt{z}-1)}\ \rangle$

```
  res = res * x;
```

$\langle \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-(\mathtt{z}-1)}\ \rangle$

```
  z = z - 1;
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

```
}
```

$\langle\ \mathtt{x} > 0 \wedge \mathtt{y} > 0 \wedge \mathtt{z} = 0 \wedge \mathtt{res} = \mathtt{x}^{\mathtt{y}-\mathtt{z}}\ \rangle$

$\langle\ \mathtt{res} = \mathtt{x}^{\mathtt{y}}\ \rangle$

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



(b) Geben Sie (ohne Beweis) eine Variante für die `while`-Schleife an, die belegt, dass die Schleife für $\mathtt{x} > \mathtt{0} \wedge \mathtt{y} > \mathtt{0}$ terminiert.

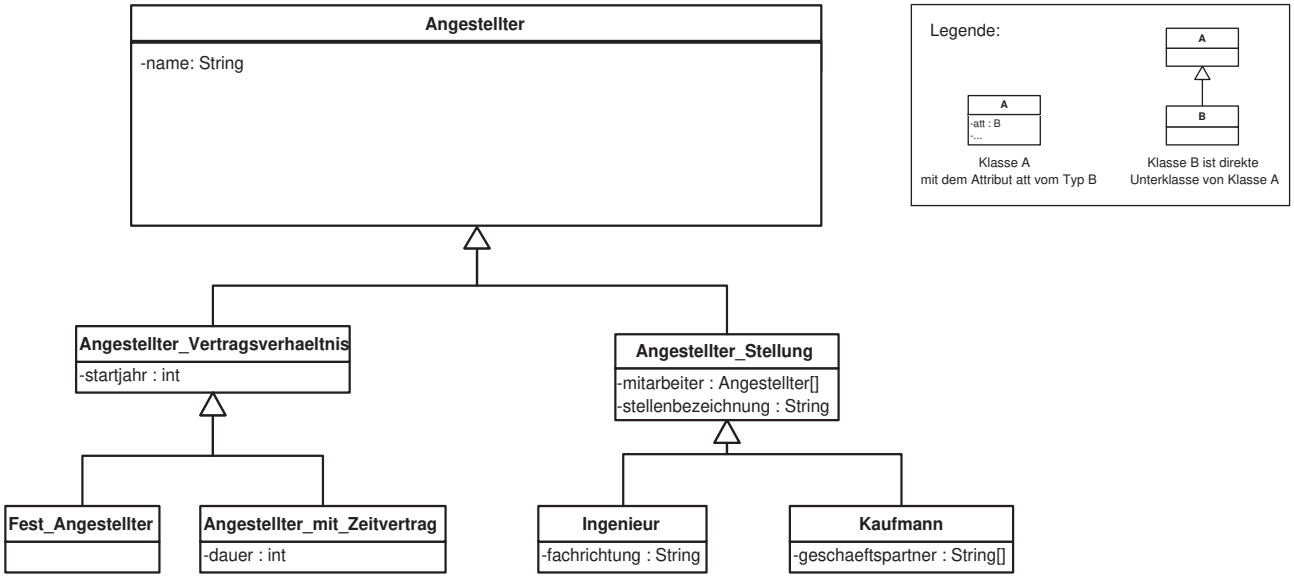
Die Variante ist $z$, da $z$ in jedem Schleifendurchlauf um `1` verkleinert wird.

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



## Aufgabe 3 (Datenstrukturen in Java, 2 + 3 + 7 Punkte)

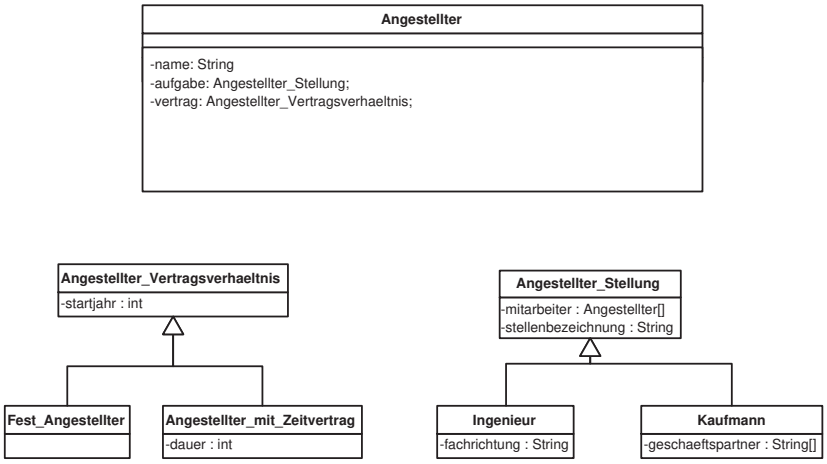
In dieser Aufgabe sollen unterschiedliche Angestelltenverhältnisse modelliert werden. Einen ersten (naiven) Entwurf zeigt die folgende Abbildung:




(a) Erläutern Sie, warum der oben angegebene Entwurf eine schlechte Modellierung der Angestelltenverhältnisse ist.

Das Problem ist, dass `Angestellter_Vertragsverhaeltnis` und `Angestellter_Stellung` als Unterklassen von `Angestellter` modelliert wurden. Auf diese Weise kann man z.B. nicht ausdrücken, dass ein Ingenieur einen Zeitvertrag hat.

(b) Verbessern Sie den obigen Entwurf sinnvoll (durch Veränderung im obigen Klassendiagramm).




Alternativ reicht es auch, nur eine der beiden Klassen `Angestellter_Vertragsverhaeltnis` oder `Angestellter_Stellung` in ein Attribut anstelle einer Unterklasse umzuwandeln.

(c) Die Klasse `Angestellter` soll eine Methode

```
public void drucke(int jahr) {...}
```

enthalten, die den Namen des Angestellten, die Anzahl seiner Mitarbeiter und sein Gehalt im Jahr

`jahr` auf dem Bildschirm ausgibt.

Das Gehalt eines Angestellten bestimmt sich nach der Anzahl der Jahre, die er bereits angestellt ist. Im ersten Jahr verdient ein Angestellter 1000 Euro, im zweiten Jahr 2000 Euro, etc. Angestellte werden immer zu Beginn des Jahres eingestellt und bei Zeitverträgen gibt `dauer` die Vertragsdauer in Jahren an.

Implementieren Sie die Klasse `Angestellter` mit der Methode `drucke`. Hierzu dürfen Sie natürlich beliebige Hilfsmethoden (auch in anderen Klassen) schreiben. Geben Sie auch die Implementierungen von denjenigen anderen Klassen an, in denen Sie Hilfsmethoden schreiben. Implementieren Sie keine Konstruktoren. Sie brauchen auch nur diejenigen Selektoren zu implementieren, die Sie für die Methode `drucke` benötigen. Verwenden Sie soweit wie möglich Vererbungstechniken und Prinzipien der Datenkapselung.

```
public class Angestellter {
    private String name;
    private Angestellter_Stellung aufgabe;
    private Angestellter_Vertragsverhaeltnis vertrag;

    public void drucke(int jahr) {
        System.out.println(name + " hat " + aufgabe.anzahlMitarbeiter() +
                    " Mitarbeiter und verdient " + vertrag.gehalt(jahr));
    }
}
public class Angestellter_Stellung {
    private Angestellter[] mitarbeiter;

    public int anzahlMitarbeiter() {
        if   (mitarbeiter != null)
             return mitarbeiter.length;
        else return 0;
    }
}
public class Angestellter_Vertragsverhaeltnis {
    protected int startjahr;

    public int gehalt(int jahr) {
        int beschaeftigungsdauer = jahr - startjahr + 1;
        if   (beschaeftigungsdauer > 0)
             return beschaeftigungsdauer * 1000;
        else return 0;
    }
}
public class Zeitvertrag extends Angestellter_Vertragsverhaeltnis {
        private int dauer;

        public int gehalt(int jahr) {
            if   (startjahr + dauer - 1 < jahr)
                 return 0;
            else return super.gehalt(jahr);
        }
}
```

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



## Aufgabe 4 (Programmierung in Java, 1 + 6 + 3 + 7 + 2 Punkte)

Der Surfverband möchte die Rangliste seiner Surfer elektronisch verwalten. Surfer sind Sportler und für die Verwaltung von Sportlern in Ranglisten existieren bereits folgende Vorgaben für die Implementierung:

```
public abstract class Sportler {
  protected String name;
}

public abstract class Element {
  protected Sportler wert;
  protected Element next;
}

public abstract class Rangliste {
  protected Element kopf;
  public Rangliste (Element kopf) {this.kopf = kopf;}
  public abstract Element sucheSportler(Sportler s);
  public abstract void aktualisiere(Sportler gewinner, Sportler verlierer);
}
```

Da beliebig viele Sportler in die jeweilige Rangliste aufgenommen werden sollen, kann die Länge der Rangliste nicht vorher festgelegt werden. Für die Rangliste sind hier nur diejenigen Methoden aufgeführt, die für das Aktualisieren der Liste aufgrund eines Spielergebnisses benötigt werden. In einem Wettkampf treten immer zwei Sportler gegeneinander an und es gibt immer einen Gewinner und einen Verlierer. Nach jedem Wettkampf wird die Rangliste aktualisiert.

Ziel der Aufgabe ist es, (nicht-abstrakte) Unterklassen `Surfer`, `SurferElement` und `SurferRangliste` von `Sportler`, `Element` und `Rangliste` zu implementieren. Hierbei dürfen die oben angegebenen Implementierungen nicht verändert werden.

Ein Surfer zeichnet sich durch einen Namen und eine Surfernummer aus. Sie dürfen davon ausgehen, dass diese Nummer eindeutig ist und brauchen dies in der Implementierung nicht zu überprüfen.

Beachten Sie auch die Sichtbarkeiten der Attribute und Methoden, da die Daten gekapselt werden sollen. Selbstverständlich dürfen Sie beliebig viele geeignete Hilfsmethoden schreiben. Sie brauchen nur die Selektoren zu implementieren, die Sie benötigen.

Gehen Sie bei der Implementierung der Klasse `SurferRangliste` wie folgt vor. Die beschriebenen Methoden können in späteren Aufgabenteilen jeweils als gegeben vorausgesetzt und verwendet werden.

(a) Implementieren Sie einen geeigneten Konstruktor

```
public SurferRangliste (Element kopf) {...}
```

in der Klasse `SurferRangliste`.

(b) Implementieren Sie die Methode `sucheSportler`. Beachten Sie: Diese Methode und die von ihr aufgerufenen Methoden dürfen keine Schleifen enthalten, sondern es darf nur *Rekursion* verwendet werden! Hierbei soll die Methode `sucheSportler (Sportler s)` in der Klasse `SurferRangliste` dasjenige Listenelement zurückliefern, welches den übergebenen Surfer `s` enthält. In der gesamten Aufgabe werden zwei Surfer als gleich betrachtet, wenn ihre Surfernummern gleich sind. Wenn `s` kein Surfer ist oder nicht in der Liste enthalten ist, soll `null` zurückgegeben werden.

Gehen Sie in der gesamten Aufgabe davon aus, dass ein Objekt der Klasse `SurferRangliste` nur `Surfer` und keine anderen `Sportler` enthält. Außerdem dürfen Sie generell voraussetzen, dass eine `SurferRangliste` keinen `Surfer` mehrmals enthält.

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



(c) Implementieren Sie eine Hilfsmethode

```
private boolean dahinterPlatziert(Surfer a, Surfer b) {...}
```

in der Klasse `SurferRangliste`, die prüft, ob der Surfer `a` in der Rangliste hinter Surfer `b` liegt. Setzen Sie dabei voraus, dass beide Surfer in der Rangliste sind.

(d) Die Methode `aktualisiere (Sportler gewinner, Sportler verlierer)` in der Klasse `SurferRangliste` soll die Rangliste wie folgt aktualisieren: Falls der Surfer `gewinner` bisher hinter dem Surfer `verlierer` in der Rangliste platziert war, wird nun der Surfer `gewinner` in der Rangliste direkt vor dem Surfer `verlierer` platziert. Wenn der Surfer `gewinner` ohnehin schon vor dem Surfer `verlierer` platziert war, so soll sich die Rangliste nicht ändern. Wenn mindestens einer der übergebenen Sportler `gewinner` bzw. `verlierer` nicht in der Rangliste oder kein Surfer ist, so soll sich die Rangliste nicht ändern.

Sie können dabei auf die Methoden `sucheSportler`, `dahinterPlatziert` und eine weitere Methode

```
public SurferElement sucheVorgaenger(Surfer s) {...}
```

in der Klasse `SurferRangliste` zurückgreifen, wobei Sie die Methode `sucheVorgaenger` *nicht* implementieren müssen. Sie gibt das Listenelement zurück, welches vor dem übergebenen Surfer `s` in der Rangliste steht. Wenn es kein solches Listenelement gibt, wird `null` zurückgegeben.

(e) Vergessen Sie nicht, die Klassen `Surfer` und `SurferElement` zu implementieren. Dabei brauchen Sie nur die Methoden (und Konstruktoren) zu schreiben, die Sie in den anderen Aufgabenteilen benutzen.

```
public class Surfer extends Sportler {

    private int surfernummer;

    public boolean equals(Sportler s) {

        if   (s instanceof Surfer)
             return (surfernummer == ((Surfer)s).surfernummer);
        else return false;
    }

}
```

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



```
public class SurferElement extends Element {

    public SurferElement getNext() {

        if   (next instanceof SurferElement)
             return (SurferElement) next;
        else return null;
    }

    public void setNext(SurferElement next) {

        this.next = next;
    }

    public SurferElement sucheSurfer(Surfer s) {

        if      (s.equals(wert))      return this;
        else if (getNext() == null)   return null;
        else                          return getNext().sucheSurfer(s);
    }

}



public class SurferRangliste extends Rangliste {

    public SurferRangliste (Element kopf) {

        super(kopf);
    }


    public Element sucheSportler(Sportler s) {

        SurferElement skopf = (SurferElement) kopf;

        if   (s instanceof Surfer && skopf != null)
             return skopf.sucheSurfer((Surfer) s);

        else {
             System.out.println("Liste leer oder kein Surfer");
             return null;
        }
    }


    private boolean dahinterPlatziert(Surfer a, Surfer b) {

        SurferElement aElement = (SurferElement) sucheSportler(a);
        return (aElement.sucheSurfer(b) == null);
    }
```

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



```
    public void aktualisiere(Sportler gewinner, Sportler verlierer) {

        Surfer        gewinnerSurfer, verliererSurfer;

        SurferElement gewinnerElement, verliererElement,
                      vorGewinnerElement, vorVerliererElement;

        if (gewinner != null && verlierer != null &&
            gewinner instanceof Surfer && verlierer instanceof Surfer) {

            gewinnerSurfer   = (Surfer) gewinner;
            verliererSurfer  = (Surfer) verlierer;
            gewinnerElement  = (SurferElement) sucheSportler(gewinner);
            verliererElement = (SurferElement) sucheSportler(verlierer);

            if (gewinnerElement == null || verliererElement == null)
                System.out.println("Mindestens ein Surfer nicht in Rangliste");

            else {

                if   (dahinterPlatziert (verliererSurfer, gewinnerSurfer))
                     System.out.println("Keine Veraenderung in Rangliste");

                else {

                     vorGewinnerElement = sucheVorgaenger(gewinnerSurfer);
                     vorVerliererElement = sucheVorgaenger(verliererSurfer);

                     vorGewinnerElement.setNext(gewinnerElement.getNext());
                     gewinnerElement.setNext(verliererElement);

                     if   (vorVerliererElement == null)
                          kopf = gewinnerElement;
                     else vorVerliererElement.setNext(gewinnerElement);
                }
            }
        }

        else System.out.println("Fehlerhafte Eingabe");
    }
}
```

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



**Aufgabe 5 (Funktionale Programmierung in Haskell,** 4 + 2 + 3 + 2 + 5 Punkte**)**

(a) Die Funktionen `f`, `g`, und `h` sind wie folgt definiert:

```
f x = x : [x]

g x y = g y x

h x  = \y -> x ++ (y : x)
```

Geben Sie den allgemeinsten Typ von `f`, `g` und `h` an.

Der allgemeinste Typ von `f` ist `a -> [a]`, der allgemeinste Typ von `g` ist `a -> a -> b` und der allgemeinste Typ von `h` ist `[a] -> a -> [a]`.

(b) Geben Sie für die folgenden Ausdrücke jeweils das Ergebnis der Auswertung an.

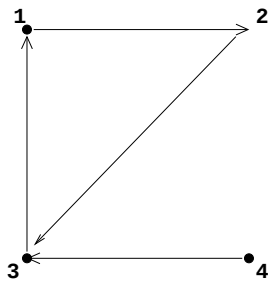
(i) `(\x -> \y -> map x y) (\x -> x + 2) [10, 3, 2]`

(ii) `filter (\x -> x < 7) (filter (\x -> x > 2) [10, 3, 2, 0, 5, 3])`

(i) `[12,5,4]`

(ii) `[3,5,3]`

(c) Graphen mit Knoten eines Typs `a` können als Liste des Typs `[(a,a)]` repräsentiert werden. Hierbei wird jede Kante des Graphens als Paar in der Liste gespeichert. Der folgende Graph




könnte also zum Beispiel durch die Liste `[ (1,2), (2,3), (3,1), (4,3) ]` repräsentiert werden.

Definieren Sie eine Funktion `delete :: a -> a -> [(a,a)] -> [(a,a)]`, wobei `delete(x,y,k)` *alle* Vorkommen der Kante von `x` nach `y` aus der Liste `k` löscht, die den Graphen repräsentiert. Wenn `k` die obige Liste ist, so ergibt `delete (2,3,k)` also das Ergebnis `[ (1,2), (3,1), (4,3)]`. Hierbei dürfen Sie keine in Haskell vordefinierten Funktionen auf Listen außer den Datenkonstruktoren `[]` und `:` verwenden. Sie dürfen davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.

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



```
delete :: a -> a -> [(a,a)] -> [(a,a)]
delete x y [] = []
delete x y (p:xs) | (x,y) == p = delete x y xs
                  | otherwise  = p : delete x y xs
```

(d) Definieren Sie in Haskell eine Datenstruktur zur Repräsentation von Graphen, welche der Beschreibung in Teil (c) entspricht. Hierbei dürfen Sie jedoch die in Haskell vordefinierten Listen *nicht* verwenden.

```
data Graph a = Empty | Edge a a (Graph a) deriving Show
```

(e) Implementieren Sie in Haskell eine Funktion `path`, die zwei Knoten `x`, `y` und einen Graph `g` als Eingabe bekommt. Dann soll `path x y g` genau dann `true` liefern, wenn es im Graph `g` einen Pfad von `x` nach `y` gibt oder wenn `x = y` ist. (Ein Pfad in einem Graphen `g` ist eine Liste $[v_1, \ldots, v_n]$ von Knoten aus `g` (mit $n \geq 0$), so dass für alle $i \in \{1, \ldots, n-1\}$ eine Kante von $v_i$ nach $v_{i+1}$ in `g` existiert. Im Beispielgraphen gibt es also einen Pfad von `1` nach `3`, aber keinen Pfad von `1` nach `4`.) Verwenden Sie hierbei Ihre Datenstruktur aus Teil (d) und geben Sie auch die Typdeklaration von `path` an. Sie dürfen wieder davon ausgehen, dass der Vergleichsoperator `(==)` für alle Typen vordefiniert ist.

```
path :: a -> a -> Graph a -> Bool
path x y Empty | x == y    = True
               | otherwise = False
path x y (Edge u v rest) = path x y rest ||
                           (path x u rest) && (path v y rest)
```

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



**Aufgabe 6 (Logische Programmierung in Prolog,** 3 + 5 + 2 + 3 Punkte**)**

(a) Definieren Sie in Prolog ein dreistelliges Prädikat `last`, welches das letzte Element von einer Liste abtrennt. Genauer bedeutet `last(X,K,L)`, dass `X` das letzte Element der Liste `L` ist und dass `K` die Liste `L` ohne ihr letztes Element ist. Beispielsweise gilt `last(3, [1,2], [1,2,3])`. Hierbei dürfen Sie keine in Prolog vordefinierten Prädikate verwenden.

```
last(X, [],    [X]  ).
last(X, [Y|K], [Y|L]) :- last(X, K, L).
```

(b) Definieren Sie ein einstelliges Prädikat `palindrom`, welches ausdrückt, dass eine Liste ein Palindrom ist.
Eine Liste $[\mathtt{A}_1, ..., \mathtt{A}_\mathtt{n}]$ ist ein Palindrom gdw. $A_j = A_{n-j+1}$ für alle $\mathtt{j} \in \{\mathtt{1}, ..., \mathtt{n}\}$. Palindrome sind z.B. `[a,b,a]`, `[a,b,b,a]`, `[a,b,c,b,a]`.

```
palindrom([]).
palindrom([X]).
palindrom([X|L]) :- last(X, K, L), palindrom(K).
```

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



(c) Formulieren Sie (basierend auf dem zuvor definierten Prädikat) eine Anfrage, um alle Palindrom-Listen zu berechnen, bei denen die ersten beiden Elemente übereinstimmen.

```
?- palindrom(L), L = [X,X|_].
```

(d) Betrachten Sie folgendes Prolog-Programm:

```
nachfolger(X,succ(X)).
nachfolger(X,Y) :- nachfolger(X,succ(Y)).
```

Terminiert das Programm bei der Suche nach den ersten drei Antworten auf die folgende Anfrage? Geben Sie ggf. die erste bzw. die ersten beiden bzw. die ersten drei Antworten an.

```
?- nachfolger(zero,U).
```

Die erste Antwort ist `U = succ(zero)`, die zweite ist `U = zero` und bei der Suche nach der dritten Antwort terminiert das Programm nicht.