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

Prof. Dr. Jürgen Giesl
Carsten Kern, Peter Schneider-Kamp, René Thiemann

Diplomvorprüfung / Zwischenprüfung
Informatik I - Programmierung
18. 9. 2006

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 oder 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 **nicht bestanden** bewertet.
- Geben Sie bitte am Ende der Klausur **alle Blätter zusammen mit den Aufgabenblättern ab**.

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	14	
Aufgabe 2	13	
Aufgabe 3	13	
Aufgabe 4	26	
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 {
    public static int x = 0;
    public A() {
        x++;
    }
    public void f(A a) {
        x++;
    }
    public void f(B a) {
        x--;
    }
}

public class B extends A {
    public int y = 2;
    public void f(A a) {
        x += y;
    }
}

public class M {
    public static void main(String[] args) {
        A a = new A();
        System.out.println(a.x);              // OUT: 1
        a.f(a);
        System.out.println(a.x);              // OUT: 2
        a = new A();
        System.out.println(a.x);              // OUT: 3
        a = new B();
        System.out.println(a.x+" "+((B)a).y); // OUT: 4 2
        a.f(a);
        System.out.println(a.x);              // OUT: 6
        B b = new B();
        System.out.println(a.x);              // OUT: 7
        b.f(b);
        System.out.println(b.x);              // OUT: 6
    }
}
```

Vorname	Name	Matr.-Nr.

3

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 static void g(A a) {
        final int z;
        int x = this.x;
        int y = a.y;
        z = x + y;
        return z;
    }
}
```

- *Die Verwendung von* `this` *ist in statischen Methoden nicht möglich.*
- *Das Objekt* `a` *ist vom Typ* `A`*. Daher hat es kein Attribut* `y`*. Dieses Attribut gibt es nur bei Objekten vom Typ* `B`*.*
- *Die Methode* `g` *ist eine* `void`*–Methode ohne Rückgabetyp. In solchen Methoden ist die Verwendung von* `return` *nicht möglich.*

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

Vorname	Name	Matr.-Nr.

4

Aufgabe 2 (Verifikation, 11 + 2 Punkte**)**

Der Algorithmus P berechnet das Quadrat einer natürlichen Zahl.

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. Sie dürfen auch weitere zusätzliche Zusicherungen einfügen.

Algorithmus: P
Eingabe: $n \in \mathbb{N}$
Ausgabe: res
Vorbedingung: $n \geq 0$
Nachbedingung: $res = n^2$

$\langle\ n \geq 0\ \rangle$

$\langle\ n \geq 0 \wedge 0 = 0^2 + 0\ \rangle$

```
res = 0;
```
$\langle\ n \geq 0 \wedge res = 0^2 + 0\ \rangle$

```
i = 0;
```
$\langle\ n \geq i \wedge res = i^2 + i\ \rangle$

```
while (n > i) {
```
$\langle\ n > i \wedge n \geq i \wedge res = i^2 + i\ \rangle$

$\langle\ n \geq i + 1 \wedge res + 2\,(i+1) = (i+1)^2 + i + 1\ \rangle$

```
  i = i + 1
```
$\langle\ n \geq i \wedge res + 2\,i = i^2 + i\ \rangle$

```
  res = res + 2*i;
```
$\langle\ n \geq i \wedge res = i^2 + i\ \rangle$

```
}
```
$\langle\ n \not> i \wedge n \geq i \wedge res = i^2 + i\ \rangle$

$\langle\ res - n = n^2\ \rangle$

```
res = res - n;
```
$\langle\ res = n^2\ \rangle$

Vorname	Name	Matr.-Nr.

5

b) Beweisen Sie die Terminierung des Algorithmus P.

Wir wählen die Variante $n - i$. Dann gilt $n > i \Rightarrow n - i \geq 0$ und

$\langle\ n - i = m \wedge n > i\ \rangle$

$\langle\ n - (i + 1) < m\ \rangle$

```
i = i + 1
```

$\langle\ n - i < m\ \rangle$

```
res = res + 2*i;
```

$\langle\ n - i < m\ \rangle$

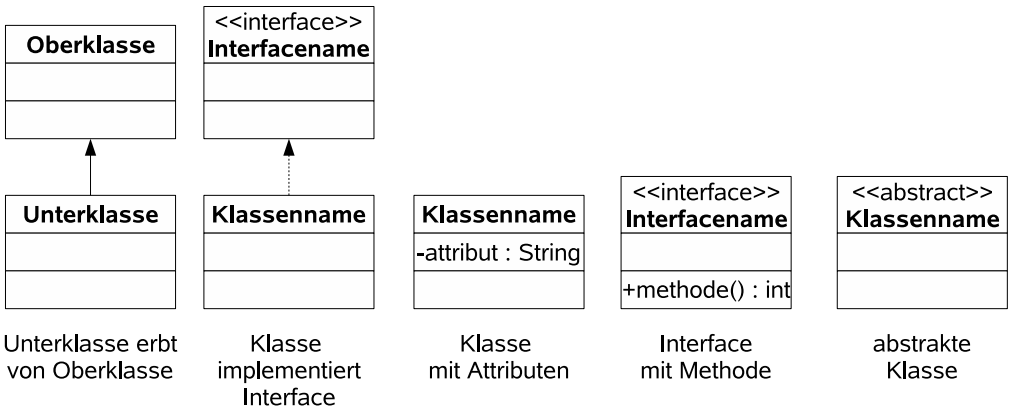
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 6 + 7 Punkte**)**

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

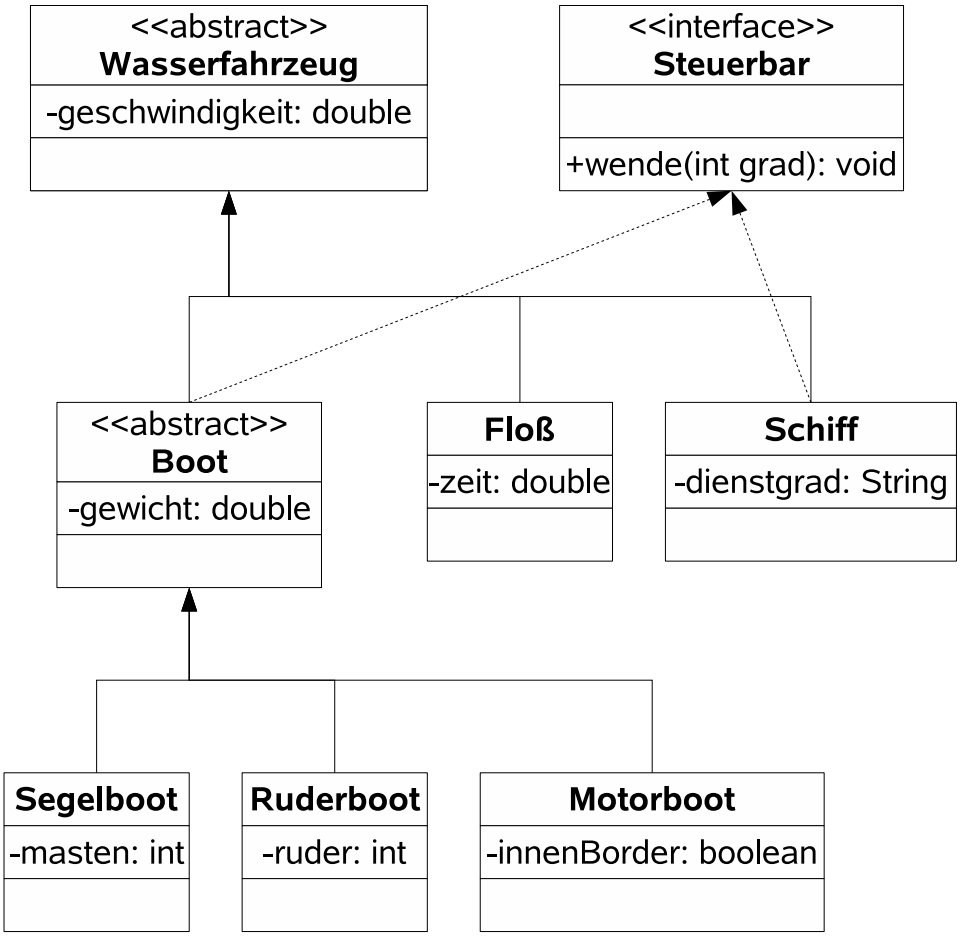
- Ein Ruderboot ist ein Wasserfahrzeug, welches auch über Land transportiert werden kann. Es verfügt über eine bestimmte Anzahl von Rudern, die die Geschwindigkeit eines Ruderboots bestimmen.
- Ein Motorboot ist ein Wasserfahrzeug mit einem Innenbord- oder Außenbord-Motor. Dieser hat Einfluss auf die Geschwindigkeit eines Motorboots. Motorboote können über Land transportiert werden.
- Ein Segelboot ist ein Wasserfahrzeug mit einer bestimmten Anzahl von Masten. Diese bedingt die Geschwindigkeit des Segelboots. Man kann Segelboote auch auf dem Landweg transportieren.
- Ein Floß ist ein Wasserfahrzeug mit einer gewissen Geschwindigkeit. Flöße zeichnen sich dadurch aus, wieviel Zeit sie noch Bestand haben, bevor sie zu morsch sind.
- Jedes Schiff ist ein Wasserfahrzeug, welches einen Kapitän mit unterschiedlich hohem Dienstgrad benötigt. Kennzeichnend für ein Schiff ist zudem die Geschwindigkeit.
- Für alle Wasserfahrzeuge außer Flößen existiert eine Methode, um sie zu steuern, indem man sie um eine gewisse Gradzahl wendet.
- Bei allen Wasserfahrzeugen, die über Land transportiert werden können (sogenannten *Booten*), wird auch ihr Gewicht gespeichert.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von Wasserfahrzeugen. 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 in der Klasse für Wasserfahrzeuge eine Methode `wartetAufMich`. Die Methode bekommt als Parameter ein Array von Wasserfahrzeugen übergeben und soll allen Wasserfahrzeugen, die schneller als das aktuelle Wasserfahrzeug sind, die Anweisung geben, um 180 Grad zu wenden, falls dieses möglich ist. Geben sie zudem auf der Konsole aus, wieviele schnellere Wasserfahrzeuge nicht umkehren können, da sie nicht steuerbar sind.

Gehen Sie dabei davon aus, dass das übergebene Array nicht der `null`-Wert ist und dass es keine `null`-Werte enthält. Kennzeichnen Sie die Methode mit dem Schlüsselwort "`static`", falls angebracht.

```
1   public void wartetAufMich(Wasserfahrzeug[] wfze) {
2     int anzahl = 0;
3     for (int i=0; i<wfze.length; i++) {
4       if (wfze[i].geschwindigkeit > this.geschwindigkeit) {
5         if (wfze[i] instanceof Steuerbar) {
6           ((Steuerbar) wfze[i]).wende(180);
7         } else {
8           anzahl++;
9         }
10      }
11    }
12    System.out.println(anzahl + " schnellere Wasserfahrzeuge"
13                       + "koennen nicht umkehren!");
14  }
```

Vorname	Name	Matr.-Nr.

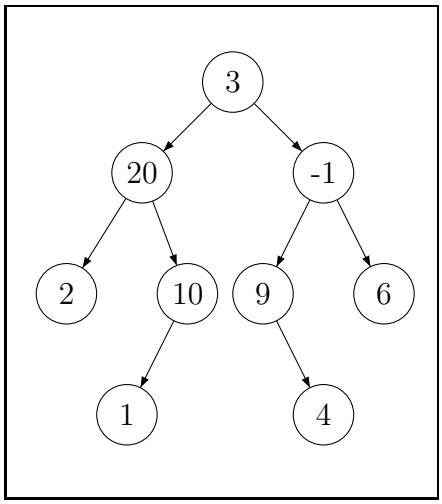
Aufgabe 4 (Programmierung in Java, 6 + 9 + 11 Punkte)

Gegeben ist das folgende Interface.

```
public interface Vergleichbar {
    public boolean gleich(Vergleichbar other); }
```

Für zwei Objekte `s` und `t` vom Typ `Vergleichbar` liefert `s.gleich(t)` den Wert `true` zurück, falls `s` gleich `t` ist und ansonsten `false`.

In dieser Aufgabe betrachten wir einen unsortierten Binärbaum. Jeder Knoten enthält hierbei einen Wert `wert` vom Typ `Vergleichbar` und hat Verweise auf seine Nachfolger (`links` und `rechts`). Das folgende Bild zeigt einen solchen Baum schematisch:



Die Datenstruktur sei folgendermaßen in Java implementiert:

```
public class Baum {
    private Knoten wurzel;

    ...
}


public class Knoten {
    Vergleichbar wert;
    Knoten links, rechts;
}
```

Vorname	Name	Matr.-Nr.

10

Implementieren Sie die folgenden Methoden in der Klasse `Baum`. Dabei dürfen Sie direkt (ohne Verwendung von Selektoren) auf die Attribute der Klasse `Knoten` zugreifen. Selbstverständlich dürfen Sie auch weitere benötigte Hilfsmethoden implementieren. Verwenden Sie die Schlüsselworte `public` und `private` auf sinnvolle Weise und kennzeichnen Sie Methoden als `static`, falls angebracht. Sie dürfen davon ausgehen, dass der `wert` eines `Knoten`s nie `null` ist.

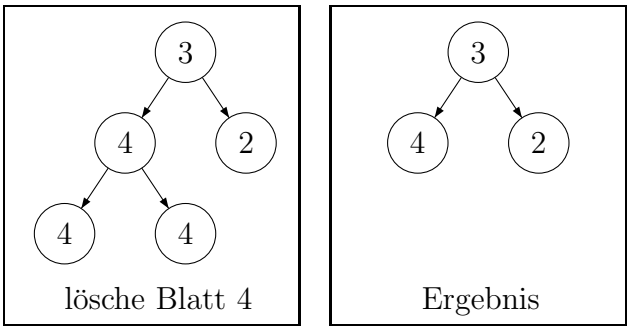
a) Implementieren Sie eine Methode `public boolean vergleicheMitBaum(Baum tree)`, die den aktuellen Baum strukturell mit einem Baum `tree` vergleicht, d.h., die Methode liefert `true` zurück, falls der aktuelle Baum und der als Parameter gegebene Baum `tree` strukturell gleich sind. Sie dürfen davon ausgehen, dass `tree` nicht der Wert `null` ist.

Zwei Bäume b_1 und b_2 heißen *strukturell gleich*, wenn ihre Wurzelknoten den gleichen Wert enthalten und der linke Unterbaum von b_1 strukturell gleich dem linken Unterbaum von b_2 sowie der rechte Unterbaum von b_1 strukturell gleich dem rechten Unterbaum von b_2 ist.

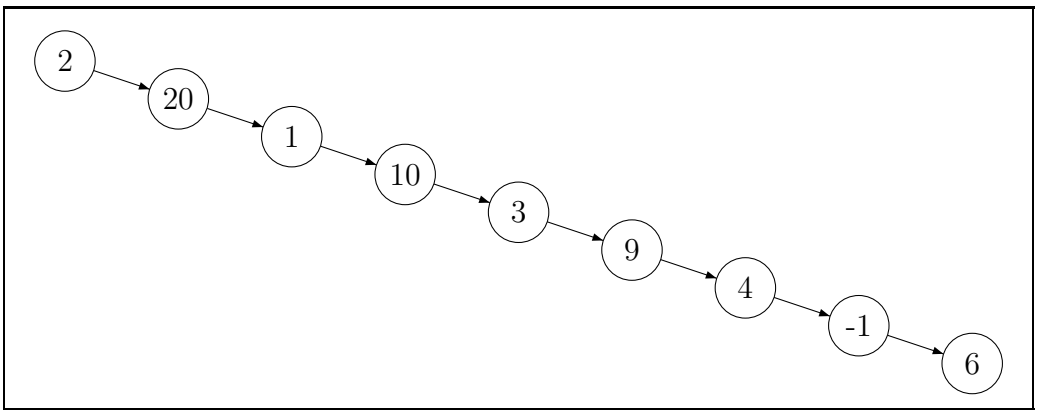
b) Implementieren Sie eine Methode `public void loescheBlaetter(Vergleichbar wert)`, die aus dem aktuellen Baum alle Blätter mit dem Wert `wert` löscht, falls solche existieren. Ansonsten werden keine Blätter gelöscht. Wie üblich ist ein *Blatt* ein Knoten des Baums, der keine Nachfolger hat. Sie dürfen davon ausgehen, dass `wert` hierbei nicht `null` ist.

Beachten Sie, dass bei der Implementierung dieser Methode keine Schleifen verwendet werden dürfen. Sie dürfen aber Rekursion benutzen.

Im folgenden Schaubild ist das Löschen aller Blätter mit Wert 4 dargestellt.



c) Implementieren Sie eine Methode `public void linearisiere ()`, die den aktuellen Baum in einen *linearisierten* Baum überführt. Ein *linearisierter* Baum ist ein Baum, bei dem kein Knoten einen linken Nachfolger hat. Die Reihenfolge der Knoten im resultierenden linearisierten Baum soll so sein, dass für jeden Knoten des ursprünglichen Baums gilt: Zuerst kommen die Werte des linken Unterbaums, dann der Wert des Knotens, dann die Werte im rechten Unterbaum. Der Baum auf der vorigen Seite wird also in folgenden linearisierten Baum überführt:



Vorname	Name	Matr.-Nr.


```
1   private static boolean vergleicheMitBaum(Knoten k1, Knoten k2){
2       if (k1 != null && k2 != null){
3           return k1.wert.gleich(k2.wert) &&
4                  vergleicheMitBaum(k1.links, k2.links) &&
5                  vergleicheMitBaum(k1.rechts, k2.rechts);
6       } else return k1 == k2;
7   }
8
9   public boolean vergleicheMitBaum(Baum tree){
10      return vergleicheMitBaum(this.wurzel, tree.wurzel);
11  }
12
13
14  private static Knoten loescheBlaetter(Knoten current, Vergleichbar wert){
15      if (current == null)
16         return null;
17      else {
18          if (current.links == null && current.rechts == null)
19              return current.wert.gleich(wert) ? null : current;
20          else{
21              current.links = loescheBlaetter(current.links, wert);
22              current.rechts = loescheBlaetter(current.rechts, wert);
23              return current;
24          }
25      }
26  }
27
28  public void loescheBlaetter(Vergleichbar wert){
29      this.wurzel = loescheBlaetter(this.wurzel, wert);
30  }
31
32
33  public static Knoten linearisiere (Knoten current) {
34      if (current == null)
35          return null;
36      else {
37          current.rechts = linearisiere(current.rechts);
38          if (current.links == null)
39              return current;
40          else {
41              Knoten linLinks = linearisiere(current.links);
42              current.links = null;
43              Knoten k;
44              for (k = linLinks; k.rechts != null; k = k.rechts);
45              k.rechts = current;
46              return linLinks;
47          }
48      }
49  }
50
51  public void linearisiere () {
52      this.wurzel = linearisiere(this.wurzel);
53  }
```

Vorname	Name	Matr.-Nr.

Aufgabe 5 (Funktionale Programmierung in Haskell, 3 + 2 + 2 + 3 + 7 Punkte)

a) Geben Sie den allgemeinsten Typ der Funktionen `f` und `g` an, die wie folgt definiert sind. Gehen Sie davon aus, dass `3` den Typ `Int` hat.

```
f = \x -> f (f x)

g x y = [y, x 3]
```

```
f :: a -> a

g :: (Int -> a) -> a -> [a]
```

b) Bestimmen Sie das Ergebnis der Auswertung für die beiden folgenden Ausdrücke. Die vordefinierte Funktion `length` vom Typ `[a]` $\rightarrow$ `Int` berechnet die Länge einer Liste.

Hinweis: Strings sind in Haskell Listen von Zeichen, d.h. es gilt `"hi"== ['h','i']`.

```
filter (\x -> x > 3) (map length [ "two", "three", "four", "five" ])
```

```
let f = \x -> if x < 2 then x == 0 else f (x-2) in f 1001
```

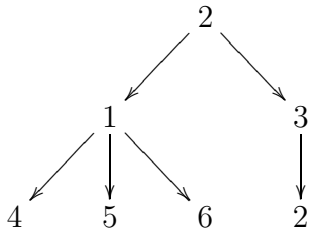
Der erste Ausdruck wertet zu `[5,4,4]` *aus und der zweite wertet zu* `False` *aus.*

c) Schreiben Sie eine Funktion `fromTo`, die zwei ganze Zahlen `n` und `m` als Eingabe bekommt und als Ergebnis die Liste der Zahlen von `n` bis `m` in aufsteigender Reihenfolge liefert. Für Zahlen `n`, die größer als `m` sind, soll `fromTo` die leere Liste als Ergebnis liefern. Beispielsweise gilt `fromTo 2 5 = [2,3,4,5]` und `fromTo 5 2 = []`. Sie dürfen in dieser Aufgabe keine vordefinierten Funktionen außer den arithmetischen Operatoren wie z.B. `+`, `-`, `>` und `==` verwenden.

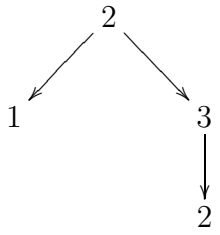
```
fromTo n m | n > m     = []
           | otherwise = n : fromTo (n+1) m
```

Vorname	Name	Matr.-Nr.

d) Ein Vielwegbaum besteht aus Knoten und Kanten, wobei jeder Knoten beliebig viele Nachfolger haben kann. Zudem ist an jedem Knoten ein Wert gespeichert. Die folgenden beiden Bäume `b1` und `b2` sind Beispiele für Vielwegbäume.



Baum `b1`



Baum `b2`

Entwerfen Sie eine Datenstruktur für Vielwegbäume in Haskell, so dass beliebige Werte in den Knoten gespeichert werden können.

Hinweis: Es gibt keinen leeren Baum.

```
data Tree a = Node a [Tree a]
```

e) Ein Vielwegbaum, in dem ganze Zahlen gespeichert sind, kann auf folgende Art beschnitten werden: Wenn in einem Knoten `k` mit Wert `n` und Nachfolgeknoten $\{k_1, \ldots, k_m\}$ die Anzahl der Nachfolger `m` echt größer als `n` ist, dann werden alle Nachfolger von `k` entfernt. Beispielsweise entsteht bei der Beschneidung von `b1` der Baum `b2`.

Implementieren Sie eine entsprechende Funktion `prune`, die einen Baum beschneidet, und geben Sie auch den Typ von `prune` an. Sie dürfen in dieser Aufgabe vordefinierte Funktionen benutzen.

Variante 1:

```
prune :: Tree Int -> Tree Int
prune (Node x children) | length children > x = Node x []
                        | otherwise           = Node x (map prune children)
```

Variante 2:

```
prune :: Tree Int -> Tree Int
prune (Node x children) | length children > x = Node x []
                        | otherwise           = Node x (pruneList children)
pruneList :: [Tree Int] -> [Tree Int]
pruneList [] = []
pruneList (x:xs) = prune x : pruneList xs
```

Vorname	Name	Matr.-Nr.

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

Analog zur Darstellung von natürlichen Zahlen mit den Funktionssymbolen `s` und `0` lassen sich auch die ganzen Zahlen darstellen. Dazu nutzt man ebenfalls `s` und `0` für den Nachfolger einer Zahl und die Zahl 0, aber zudem gibt es noch das Funktionssymbol `p`, das den Vorgänger einer Zahl repräsentiert. Mit `s`, `0` und `p` lässt sich offensichtlich jede ganze Zahl darstellen: Beispielsweise entspricht `s(s(0))` der Zahl 2 und `p(p(0))` der Zahl -2. Allerdings sind die Darstellungen nicht mehr eindeutig, da z.B. alle Terme `s(0)`, `s(p(s(0)))` und `s(p(p(s(s(0)))))` die Zahl 1 repräsentieren. Wenn in einem Term nicht gleichzeitig `p` und `s` vorkommen, dann nennen wir den Term *normalisiert*. Beispielsweise waren im bisherigen Text alle Terme bis auf `s(p(s(0)))` und `s(p(p(s(s(0)))))` normalisiert.

a) Implementieren Sie in Prolog ein Prädikat zur Addition zweier ganzer Zahlen in der Darstellung mit `s`, `0` und `p`. Das Ergebnis der Berechnung braucht nicht normalisiert sein.

```
plus(0,X,X).
plus(s(X),Y,s(Z)) :- plus(X,Y,Z).
plus(p(X),Y,p(Z)) :- plus(X,Y,Z).
```

b) Implementieren Sie in Prolog ein Prädikat `normalize` zur Normalisierung von ganzen Zahlen. Beispielsweise sollte also `normalize(s(p(s(0))),s(0))` gelten.

Variante 1:

```
normalize(0,0).
normalize(s(X),s(s(Y))) :- normalize(X,s(Y)).
normalize(s(X),Y)       :- normalize(X,p(Y)).
normalize(s(X),s(0))    :- normalize(X,0).
normalize(p(X),p(p(Y))) :- normalize(X,p(Y)).
normalize(p(X),Y)       :- normalize(X,s(Y)).
normalize(p(X),p(0))    :- normalize(X,0).
```

Variante 2:

```
normalize(X,Y) :- norm(X,0,Y).

norm(0,X,X).
norm(p(X),0,Z)    :- norm(X,p(0),Z).
norm(p(X),s(Y),Z) :- norm(X,Y,Z).
norm(p(X),p(Y),Z) :- norm(X,p(p(Y)),Z).
norm(s(X),0,Z)    :- norm(X,s(0),Z).
norm(s(X),s(Y),Z) :- norm(X,s(s(Y)),Z).
norm(s(X),p(Y),Z) :- norm(X,Y,Z).
```

Vorname	Name	Matr.-Nr.

15

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

- `f(g(h(X)),Z,Y)` und `f(g(Y),h(h(X)),Z)`
  nicht unifizierbar wegen Occur-Failure

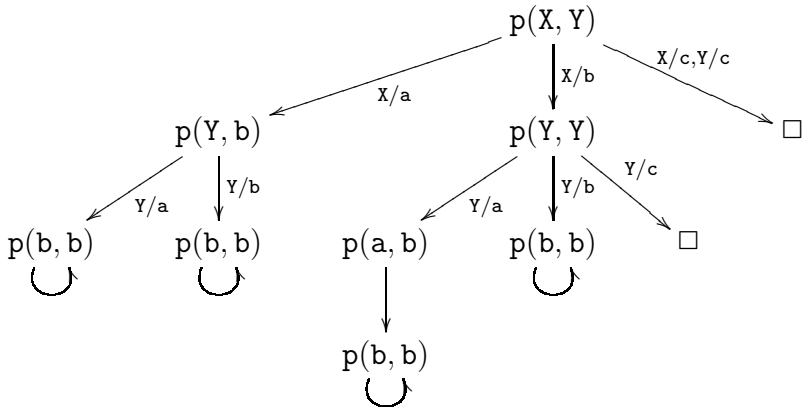
- `p(X,Y,Z)` und `p(r(Z,Z),s,r(Y,Y))`
  Allgemeinster Unifikator: `X = r(r(s,s),r(s,s)), Y=s, Z=r(s,s)`

- `u(u(X,v(a)),Y)` und `u(Y,u(v(b),X))`
  nicht unifizierbar wegen Clash-Failure

Vorname	Name	Matr.-Nr.

d) Erstellen Sie für das folgende Logikprogramm den SLD-Baum zur Anfrage "`?- p(X,Y).`" und geben Sie alle Lösungen an.

```
p(a,Z) :- p(Z,b).
p(b,Z) :- p(Z,Z).
p(c,c).
```



Hierbei steht p(b, b) *für den unendlichen Pfad aus lauter Knoten, die mit* `p(b,b)` *markiert sind.*

Die Lösungen sind `X=b, Y=c` *und* `X=c, Y=c`.

Ordnen Sie die Klauseln des Programms so an, dass Prolog alle Lösungen zur Anfrage "`?- p(X,Y).`" findet. Ist diese Anordnung eindeutig?

```
p(c,c).
p(b,Z) :- p(Z,Z).
p(a,Z) :- p(Z,b).
```

Ja, diese Reihenfolge der Klauseln ist eindeutig.