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

LuFG Informatik II

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

Prüfungsklausur Programmierung

27. 2. 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	14	
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 {
    protected static int x = 1;
    public A(int x) {
        A.x += x;
    }
    protected void f(int x) {
        A.x -= x;
    }
}

public class B extends A {
    protected double y = 3;
    public B(int x) {
        super(x);
        y++;
    }
    protected void f(int x) {
        A.x += x;
    }
    protected void f(double x) {
        y += x;
    }
}

public class M {
    public static void main(String[] args) {
        A a = new A(A.x);
        System.out.println(a.x);          // OUT: 2
        a.f(10);
        System.out.println(a.x);          // OUT: -8
        B b = new B(10);
        System.out.println(b.x+" "+b.y); // OUT: 2 4.0
        b.f(1.0);
        System.out.println(b.x+" "+b.y); // OUT: 2 5.0
        b.f(10);
        System.out.println(b.x+" "+b.y); // OUT: 12 5.0
        a = b;
        a.f(10);
        System.out.println(a.x);          // OUT: 22
    }
}
```

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 {
    private int y = 5;
    public C() {
        super(7.0);
        y++;
    }
    public int f(int x) {
        final int z = 6;
        z += x;
        return 2*x;
    }
}
```

- *Es gibt keinen Konstruktor* `B(double)`*, der von* `C()` *aufgerufen werden könnte..*
- *Die Methode* `f(int)` *in* `B` *hat den Rückgabetyp* `void`*, der beim Überschreiben nicht auf* `int` *geändert werden darf.*
- *Die Variable* `z` *ist als* `final` *deklariert und darf deshalb nur einmal gesetzt werden.*

Vorname	Name	Matr.-Nr.

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

Der Algorithmus P berechnet für eine Zahl $n \in \mathbb{N} = \{0, 1, 2, \dots\}$ ihre Fakultät $n! = \prod_{i=1}^{n} i$. Generell gilt $\prod_{i=k}^{m} i = 1$, falls $k > m$. Somit ist also $0! = 1$.

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: $n \in \mathbb{N} = \{0, 1, 2, \dots\}$
Ausgabe: res
Vorbedingung: $n \geq 0$
Nachbedingung: $res = \prod_{i=1}^{n} i$

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

$\langle\ n \geq 0 \wedge 1 = 1\ \rangle$

$res = 1;$

$\langle\ n \geq 0 \wedge res = 1\ \rangle$

$\langle\ n \geq 0 \wedge res = 1 \wedge 0 = 0\ \rangle$

$p = 0;$

$\langle\ n \geq p \wedge res = 1 \wedge p = 0\ \rangle$

$\langle\ n \geq p \wedge res = \prod_{i=n-p+1}^{n} i\ \rangle$

`while` $(n > p)$ {

$\langle\ n \geq p \wedge res = \prod_{i=n-p+1}^{n} i \wedge n > p\ \rangle$

$\langle\ n \geq p + 1 \wedge res * (n - p) = (n - p) * \prod_{i=n-p+1}^{n} i\ \rangle$

$res = res * (n - p);$

$\langle\ n \geq p + 1 \wedge res = (n - p) * \prod_{i=n-p+1}^{n} i\ \rangle$

$\langle\ n \geq p + 1 \wedge res = \prod_{i=n-(p+1)+1}^{n} i\ \rangle$

$p = p + 1;$

$\langle\ n \geq p \wedge res = \prod_{i=n-p+1}^{n} i\ \rangle$

}

$\langle\ n \geq p \wedge res = \prod_{i=n-p+1}^{n} i \wedge n \not> p\ \rangle$

$\langle\ res = \prod_{i=1}^{n} i\ \rangle$

Vorname	Name	Matr.-Nr.

b) Beweisen Sie die Terminierung des Algorithmus P.

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

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

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

$res = res * (n - p);$

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

$p = p + 1;$

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

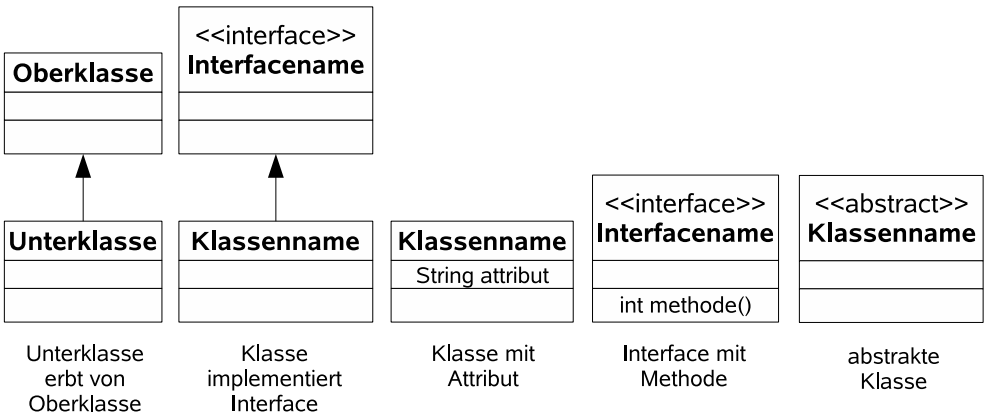
Vorname	Name	Matr.-Nr.

Aufgabe 3 (Datenstrukturen in Java, 6 + 8 Punkte**)**

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

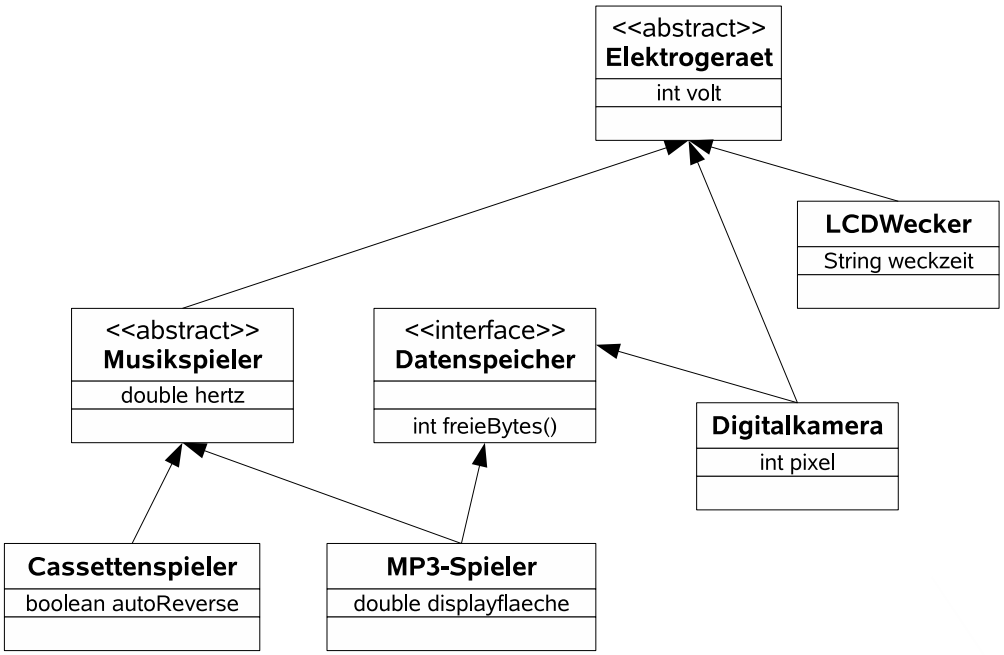
- Jedes elektronische Gerät wird durch seine Betriebsspannung in Volt gekennzeichnet.
- Ein LCD-Wecker ist ein elektronisches Gerät, das sich insbesondere durch die eingestellte Weckzeit auszeichnet.
- Eine Digitalkamera ist ein elektronisches Gerät, bei dem die Anzahl an Pixeln des Bildsensors eine wichtige Kenngröße ist.
- Ein Musikspieler ist ein elektronisches Gerät, für das die maximale Frequenz in Hertz interessant ist, die der Musikspieler wiedergeben kann.
- Ein Cassettenspieler ist ein Musikspieler, wobei manche Cassettenspieler auch über die so genannte „Auto Reverse“-Funktionalität verfügen (damit kann solch ein Cassettenspieler beide Seiten einer Musikcassette wiedergeben, ohne dass der Benutzer sie per Hand umdrehen muss).
- Ein MP3-Spieler ist ein Musikspieler, wobei für MP3-Spieler die Fläche des Displays eine wichtige Angabe ist.
- Sowohl MP3-Spieler als auch Digitalkameras können an einen Computer angeschlossen werden und zur Speicherung von Daten verwendet werden. Deshalb stellen sie eine Methode zur Verfügung, mit der sich der aktuell verfügbare Speicherplatz in Bytes bestimmen lässt.

a) Entwerfen Sie unter Berücksichtigung der Prinzipien der Datenkapselung eine geeignete Klassenhierarchie für die oben aufgelisteten Arten von elektronischen Geräten. 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 und Datentypen ihrer Attribute an. Methoden von Klassen müssen nicht angegeben werden. Geben Sie für jedes Interface ausschließlich den jeweiligen Namen sowie die Namen und Ein- und Ausgabetypen seiner Methoden an.

Vorname	Name	Matr.-Nr.



b) Implementieren Sie in Java eine Methode `freieKamera`. Die Methode bekommt als Parameter ein Array von elektronischen Geräten übergeben. Sie soll die erste Digitalkamera im übergebenen Array zurückgeben, die noch genügend freien Speicherplatz für ein Bild hat. Gehen Sie dabei davon aus, dass ein Bild mit n Pixeln genau $3n$ Bytes an Speicherplatz belegt. Falls keine solche Digitalkamera im übergebenen Array vorhanden ist, soll der Wert `null` zurückgegeben werden.

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 Digitalkamera freieKamera(Elektrogeraet[] array) {
    for (int i = 0; i < array.length; i++) {
        Elektrogeraet eGeraet = array[i];
        if (eGeraet instanceof Digitalkamera) {
            Digitalkamera digicam = (Digitalkamera) eGeraet;
            if (digicam.freieBytes() >= 3*digicam.getPixel()) {
                return digicam;
            }
        }
    }
    return null;
}
```

Vorname	Name	Matr.-Nr.

Aufgabe 4 (Programmierung in Java, 8 + 11 + 7 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.

10

a) Implementieren Sie die Methode

```
public Liste konkateniere(Liste ys)
```

der Klasse `Liste`. Diese Methode bekommt eine Liste `ys` als Eingabeargument und soll eine *neue* Liste zurückliefern, die erst alle Elemente der aktuellen Liste und dann alle Elemente der Eingabeliste enthält. Sie dürfen hierbei davon ausgehen, dass die Eingabeliste `ys` nicht `null` ist. Falls die aktuelle Liste $[x_1, \ldots, x_n]$ ist, und `ys` die Liste $[y_1, \ldots, y_m]$ ist, so sollte `konkateniere(ys)` also die *neue* Liste $[x_1, \ldots, x_n, y_1 \ldots, y_m]$ zurückliefern. (Diese Liste soll also insbesondere andere `Element`-Objekte als die Ursprungslisten enthalten und die aktuelle Liste und die Eingabeliste `ys` sollten hierbei nicht verändert werden.)

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 konkateniere(Liste ys){
     return new Liste(konkateniere(this.start,ys.start));
}

private static Element konkateniere(Element a, Element b) {
    if (a == null && b == null) return null;
    if (a == null){
        return new Element(b.wert,konkateniere(a,b.nachfolger));
    }
    return new Element(a.wert,konkateniere(a.nachfolger,b));
}
```

b) Das Interface `Verknuepfung` hat eine Methode `verknuepfe`, die aus zwei Zahlen eine neue Zahl berechnet.

```
public interface Verknuepfung {

    public int verknuepfe(int wert1, int wert2);

}
```

Implementieren Sie die Methode

`public Liste verknuepfeListen(Liste ys, Verknuepfung v)`

der Klasse `Liste`. Sie soll aus der aktuellen Liste und der Eingabeliste `ys` eine neue Liste berechnen. Das erste Element der neuen Liste ergibt sich, indem man `v.verknuepfe` auf das erste Element der aktuellen Liste und das erste Element der Liste `ys` anwendet. Das zweite Element der neuen Liste ergibt sich, indem man `v.verknuepfe` auf das zweite Element der aktuellen Liste und das zweite Element von `ys` anwendet, etc. Sie können davon ausgehen, dass die Eingabeliste `ys` nicht `null` ist und dass die aktuelle Liste und die Eingabeliste `ys` gleich lang sind. Falls die aktuelle Liste also $[\mathtt{x}_1, \ldots, \mathtt{x}_n]$ ist und `ys` die Liste $[\mathtt{y}_1, \ldots, \mathtt{y}_n]$ ist, so sollte `verknuepfeListen(ys,v)` die neue Liste $[\mathtt{v.verknuepfe}(\mathtt{x}_1, \mathtt{y}_1), \ldots, \mathtt{v.verknuepfe}(\mathtt{x}_n, \mathtt{y}_n)]$ zurückliefern. Die aktuelle Liste und die Eingabeliste `ys` sollten hierbei nicht verändert werden.

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 verknuepfeListen(Liste ys, Verknuepfung v){
    return new Liste(verknuepfeElement(this.start,ys.start,v));
}

private static Element verknuepfeElement(Element a, Element b, Verknuepfung v) {
    if (a == null || b == null) return null;
    int x = v.verknuepfe(a.wert, b.wert);
    return new Element(x,verknuepfeElement(a.nachfolger,b.nachfolger,v));
}
```

c) Implementieren Sie die Methode

```
public Liste addiere(Liste ys)
```

der Klasse `Liste`. Sie soll aus der aktuellen Liste und der Eingabeliste `ys` eine neue Liste berechnen. Das erste Element der neuen Liste ergibt sich, indem man das erste Element der aktuellen Liste und das erste Element der Liste `ys` addiert. Das zweite Element der neuen Liste ergibt sich, indem man das zweite Element der aktuellen Liste und das zweite Element von `ys` addiert, etc. Sie können davon ausgehen, dass die Eingabeliste `ys` nicht `null` ist und dass die aktuelle Liste und die Eingabeliste `ys` gleich lang sind. Falls die aktuelle Liste also $[x_1, \ldots, x_n]$ ist und `ys` die Liste $[y_1, \ldots, y_n]$ ist, so sollte `addiere(ys)` die neue Liste $[x_1 + y_1, \ldots, x_n + y_n]$ zurückliefern. Die aktuelle Liste und die Eingabeliste `ys` sollten hierbei nicht verändert werden.

Verwenden Sie bei Ihrer Implementierung *keine Schleifen* und auch *keine Rekursion*. Implementieren Sie stattdessen eine geeignete Klasse `Addierer`, die das Interface `Verknuepfung` implementiert. Die Methode `addiere` aus der Klasse `Liste` sollte dann die Methode `verknuepfeListen` aus Aufgabenteil b) sowie Ihre Klasse `Addierer` verwenden.

```
public class Addierer implements Verknuepfung {

    public int verknuepfe(int x, int y) {
        return x + y;
    }

}

public Liste addiere(Liste ys){
    return verknuepfeListen(ys,new Addierer());
}
```

Vorname	Name	Matr.-Nr.

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

a) Geben Sie den allgemeinsten Typ der Funktionen `f` und `g` an, die wie folgt definiert sind. Gehen Sie davon aus, dass `1` den Typ `Int` hat.

```
f = \x y -> (y ++ [x+1])

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

g e h x = h (h (e x))

g :: (a -> b) -> (b -> b) -> a -> b
```

b) Sei das folgende Programm gegeben:

```
take :: Int -> [a] -> [a]
take 0 _  = []
take _ [] = []
take n (x:xs) =  x:take (n-1) xs

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

bot :: a
bot = bot

get :: [a] -> Int -> a
get (x:xs) 1 = x
get (x:xs) n = get xs (n-1)

h :: [(Int,a)] -> Int -> [a]
h xs 0        = []
h xs n        = c:h xs m where (m,c) = get xs n
```

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

```
map (\x->x*2) (take 2 [2,4,bot])
```

`[4,8]`

```
h [(1,'y'),(4,'h'),(2,'r'),(0,'i')] 2
```

`"hi"` oder `['h','i']`

Vorname	Name	Matr.-Nr.

c) Schreiben Sie eine Funktion `reverse` in Haskell, die eine Liste umdreht. Benutzen Sie keine vordefinierten Funktionen. Geben Sie außerdem den allgemeinsten Typ der Funktion `reverse` an. Beispielsweise liefert der Aufruf `reverse [1,2,3,4]` die Liste `[4,3,2,1]` und der Aufruf `reverse [True,False]` liefert die Liste `[False,True]`.

```
reverse :: [a] -> [a]
reverse xs = reverse' xs []
reverse' :: [a] -> [a] -> [a]
reverse' [] xs = xs
reverse' (x:xs) ys = reverse' xs (x:ys)
```

d) Ein binärer Listenbaum hat Knoten, die jeweils eine Liste von Werten eines Typs speichern. Jeder Knoten hat zwei Teilbäume, wobei ein Teilbaum durchaus der leere Listenbaum sein kann. Der leere Listenbaum speichert *keine* Werte und hat *keine* Teilbäume. Zwei Beispiele für binäre Listenbäume finden Sie auf der nächsten Seite. Entwerfen Sie eine Datenstruktur für binäre Listenbäume in Haskell.

```
data BLB a = Blatt | Knoten [a] (BLB a) (BLB a)
```

Vorname	Name	Matr.-Nr.

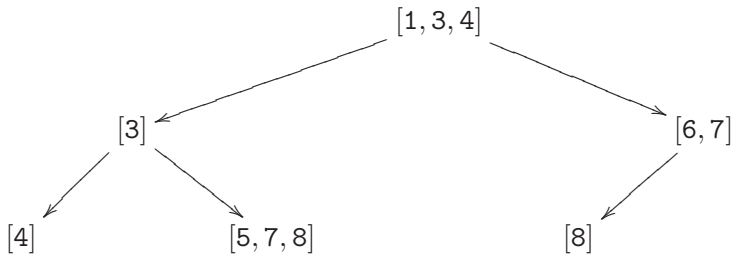


Abbildung 1: Der binäre Listenbaum `a`

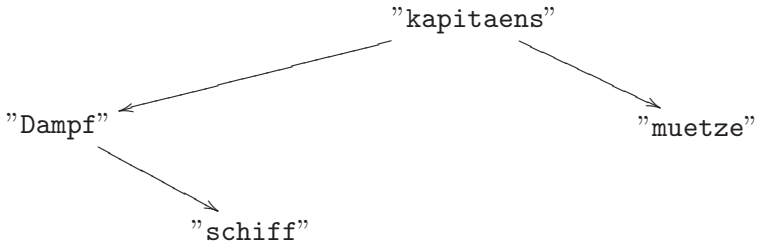


Abbildung 2: Der binäre Listenbaum `b`

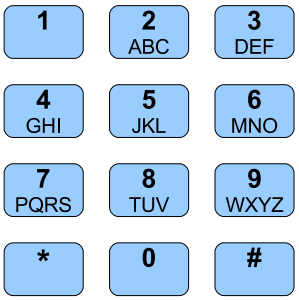
e) Schreiben Sie eine Funktion `alsListe` in Haskell, die einen binären Listenbaum in eine Liste umwandelt. Dabei soll die Ergebnisliste die Konkatenierung aller Listen im Listenbaum sein. Die Listen des Baums sollen in Links-Rechts-Reihenfolge aneinander gehängt werden. Das heißt: Für einen Listenbaum wird erst der linke Teilbaum der Wurzel in eine Liste umgewandelt. Dann wird an diese Liste die Liste des Wurzelknotens angehängt. Anschließend wird der rechte Listenbaum zu einer Liste umgewandelt und als letztes angehängt. Beispielsweise liefert der Aufruf `alsListe a` die Liste `[4,3,5,7,8,1,3,4,8,6,7]` und der Aufruf `alsListe b` liefert die Liste `"Dampfschiffkapitaensmuetze"` (Ein String ist eine Liste vom Typ `[Char]`). Geben Sie auch den allgemeinsten Typ der Funktion `alsListe` an.

```
alsListe :: (BLB a) -> [a]
alsListe Leaf= []
alsListe (Knoten xs l r) = alsListe l ++ xs ++ alsListe r
```

Vorname	Name	Matr.-Nr.

Aufgabe 6 (Logische Programmierung in Prolog, (3 + 2 + 1) + 4 + (6 + 1) Punkte**)**

a) Unser Alphabet mit 26 Buchstaben kann auf einer Telefontastatur mit nur 10 Ziffern nicht direkt dargestellt werden. Um trotzdem Namen und Kurznachrichten auf einer solchen Tastatur eingeben zu können, wurde der T9-Code eingeführt. Dieser bildet die Tasten 2 - 9 auf je drei bzw. vier Buchstaben ab. Eine solche T9-Tastatur und die Kodierung mit Hilfe von Prolog-Fakten könnten so aussehen:




```
m(2,a). m(2,b). m(2,c).
m(3,d). m(3,e). m(3,f).
m(4,g). m(4,h). m(4,i).
m(5,j). m(5,k). m(5,l).
m(6,m). m(6,n). m(6,o).
m(7,p). m(7,q). m(7,r). m(7,s).
m(8,t). m(8,u). m(8,v).
m(9,w). m(9,x). m(9,y). m(9,z).
```

- Implementieren Sie in Prolog ein zweistelliges Prädikat `tneun`, welches zu einer Liste von Ziffern (von 2 - 9) alle möglichen Wörter berechnet, die dieser Liste nach dem T9-Code entsprechen. Zum Beispiel sollen sich für die Anfrage "`?- tneun([4,8],W)`" die Lösungen `W=[g,t]`, `W=[g,u]`, `W=[g,v]`, `W=[h,t]`, `W=[h,u]`, `W=[h,v]`, `W=[i,t]`, `W=[i,u]` und `W=[i,v]` ergeben.

```
tneun([],[]).
tneun([X|Xs],[Y|Ys]) :- m(X,Y), tneun(Xs,Ys).
```

- Implementieren Sie in Prolog ein zweistelliges Prädikat `synonym`, welches für ein Wort (gegeben durch eine Liste von Buchstaben) alle *T9-Synonyme* berechnet. Hierbei sind zwei Wörter *T9-Synonyme* genau dann, wenn sie im T9-Code durch die gleiche Liste von Ziffern repräsentiert werden. Die Anfrage "?- `synonym([s,m,s],S)`" sollte also unter anderem die Lösung `S = [p,o,p]` ergeben (d.h. `S` sollte unter anderem an `[p,o,p]` gebunden werden).

```
synonym(Xs,Ys) :- tneun(Zs,Xs), tneun(Zs,Ys).
```

- Geben Sie eine Anfrage mit Hilfe des Prädikats `synonym` an, so dass alle Synonyme von `[b,o,o,k]`, die mit `c` anfangen und mit `l` aufhören, an die Variable `L` gebunden werden.

```
synonym([b,o,o,k],L), L = [c,_,_,l].
```

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,H,f(U),f(Y),b)`
  nicht unifizierbar wegen Clash-Failure:

  ```
  H = a
  U = a
  R = f(a)
  Y = a
  a = b <- CLASH FAILURE
  ```

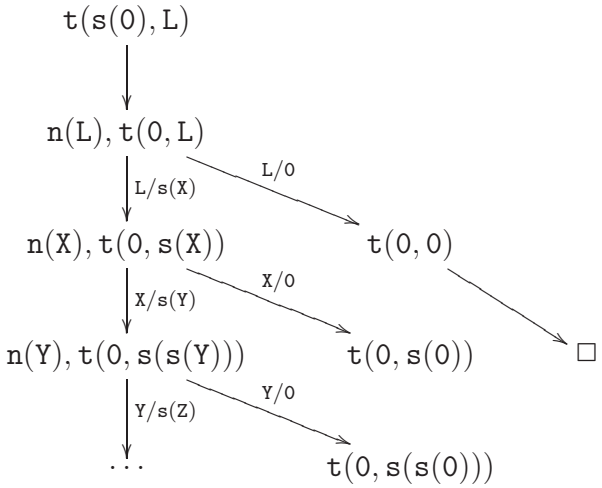
- `f(A,g(B,B),g(A,A))` und `f(g(C,C),C,B)`
  nicht unifizierbar wegen Occur-Failure:

  ```
  A = g(g(B,B),g(B,B))
  C = g(B,B)
  B = g(g(g(B,B),g(B,B)),g(g(B,B),g(B,B))) <- OCCUR FAILURE
  ```

Vorname	Name	Matr.-Nr.

c) Erstellen Sie für das folgende Logikprogramm den SLD-Baum zur Anfrage "`?- t(s(0),L).`" Sie dürfen dabei Pfade abbrechen, sobald diese eine Anfrage enthalten, in der die Funktion `s` drei mal vorkommt. Kennzeichnen Sie solche Knoten durch "...". Geben Sie weiterhin alle Lösungen an und begründen Sie, warum es keine weiteren gibt. Welche Lösungen findet Prolog?

```
n(s(X)) :- n(X).
n(0).
t(s(X),U) :- n(U), t(X,U).
t(0,0).
```



Die einzige Lösung ist `L=0` *und wird von Prolog nicht gefunden, da Prolog zuerst den unendlichen Ast untersucht. Der unendliche Ast enthält keine weiteren Lösungen, da das letzte Atom jeder Anfrage die Form* `t(0,s(...))` *besitzt. Diese Anfrag unifiziert jedoch mit keinem Klauselkopf.*

Ordnen Sie die Klauseln und Atome des Programms so an, dass Prolog bei der Anfrage "`?- t(s(0),L).`" terminiert, d.h., dass der SLD-Baum endlich wird.

```
n(0).
n(s(X)) :- n(X).
t(0,0).
t(s(X),U) :- t(X,U), n(U).
```