Probeklausur MPGI 1
01.02.2012

Prof. Dr. Glesner und WMs

Name: .....

Vorname: .....

Matr.-Nr.: .....

Bearbeitungszeit: 75 Minuten

Aufgabe	Thema	Punkte	erreicht
1	Kundendatenbank	7	
2	Listenfunktionale	9	
3	Lambda-Kalkül	2	
4	Aufwand	3	
5	Bäume	10	
6	Sortierverfahren	2	
7	Ein- und Ausgabe	5	
Summe		38	

Hinweise zur Bearbeitung der OPAL-Aufgaben

Bei der Bearbeitung der OPAL-Aufgaben kann auf `IMPORT`-, `SIGNATURE`- und `IMPLEMENTATION`-Deklarationen verzichtet werden. Es können alle Funktionen aus der OPAL-Standardbibliothek verwendet werden, soweit es nicht anders angegeben ist. Eine **nicht vollständige** Liste der Strukturen und Funktionen aus der Bibliothek ist im Folgenden angegeben.

```
SIGNATURE NatConv
-- --------------
FUN asReal : nat -> real

SIGNATURE Seq[data]
-- ----------------
SORT data
TYPE seq == <>
            ::(ft : data, rt : seq)
FUN #       : seq -> nat
FUN ++      : seq ** seq -> seq
FUN =       : (data ** data -> bool) -> seq ** seq -> bool
FUN ft last : seq -> data
FUN split   : nat ** seq -> seq ** seq
FUN +%      : seq ** data -> seq

SIGNATURE SeqMap[from,to]
-- ----------------------
SORT from to
FUN map : (from -> to) -> seq[from] -> seq[to]

SIGNATURE SeqFilter[data]
-- ----------------------
SORT data
FUN filter : (data -> bool) -> seq[data] -> seq[data]

SIGNATURE SeqReduce[from,to]
-- -------------------------
SORT from to
FUN reduce : (from ** to -> to) ** to -> seq[from] -> to

SIGNATURE SeqZip[from1,from2,to]
-- -----------------------------
SORT from1 from2 to
FUN zip : (from1 ** from2 -> to) -> seq[from1] ** seq[from2] -> seq[to]

SIGNATURE Com[data]
-- ---------------------
SORT data com
FUN succeed : data -> com

SIGNATURE BasicIO
-- --------------
FUN write     : int -> com[void]
FUN writeLine : int -> com[void]
FUN ask       : denotation -> com[int]
FUN ask       : denotation -> com[denotation]

SIGNATURE ComCompose[first,second]
-- -------------------------------
SORT first second
FUN & : com[first] ** (first -> com[second]) -> com[second]
```

1. Aufgabe (7 Punkte): Kundendatenbank

In dieser Aufgabe sollen Datentypen zur Verwaltung des Kundenstamms eines Eisverkäufers implementiert werden. Jeder Kunde hat einen Namen, ein Alter, eine Lieblingssorte und einen Umsatz aus dem Vorjahr. Der Datentyp für die Eissorten ist folgendermaßen deklariert:

```
TYPE Eissorte == Vanille Schokolade Mango Erdbeer Nuss
```

Damit der Eisverkäufer ohne Gleitkommazahlen auskommt, werden Beträge in Euro und Cent getrennt angegeben:

```
TYPE Betrag == betrag(euro : nat, cent : nat)
```

In den folgenden Aufgaben könnt ihr davon ausgehen, dass den jeweiligen Funktionen stets „korrekt gebildete“ Beträge übergeben werden, d.h. in denen der Cent-Wert zwischen 0 und 99 liegt.

1.1. Induzierte Signatur (2 Punkte) Geben Sie die induzierte Signatur des Datentyps `Kunden` an. Benennen Sie dabei die einzelnen Komponenten. Geben Sie an, um welche Art von Datentyp es sich bei `Kunden` handelt. Begründen Sie kurz Ihre Antwort.

```
TYPE Kunden == kunde (name           : denotation,
                      alter          : nat,
                      lieblingssorte : Eissorte,
                      umsatz         : Betrag)
```

1.2. Erweitern des Datentyps `Kunden` (2 Punkte) Erweitern Sie den Datentypen Kunden um einen geschäftlichen Kunden `grosskunde`, welcher den Namen einer Firma anstatt des Alters enthält. Welche Funktionen der induzierten Signatur des erweiterten Datentyps sind **total**?

```
TYPE Kunden == kunde (name           : denotation,
                      alter          : nat,
                      lieblingssorte : Eissorte,
                      umsatz         : Betrag)
```

1.3. Vergleichsoperator (1 Punkt) Deklarieren und definieren Sie eine Funktion `<`, welche zwei Beträge miteinander vergleicht und `true` zurückliefert, wenn der erste Betrag kleiner als der zweite Betrag ist, und sonst `false`.

Beispiel: betrag(10,90) $<$ betrag(4,17) $\rightsquigarrow$ false

1.4. Suchen von guten Kunden (2 Punkte) Ein Eisverkäufer will alle privaten und geschäftlichen Kunden, deren Umsatz besonders hoch war, kennen. Deklarieren und definieren Sie eine Funktion `guteKunden`, die aus einer Liste von Kunden, die **Namen** aller Kunden als Sequenz von `denotation` zurückgibt, deren Umsatz größer als 24,50 bzw. bei geschäftlichen Kunden 1000 Euro ist. Benutzen Sie dabei den Vergleichsoperator aus der vorherigen Aufgabe.

Schreiben Sie **rekursive Funktionen**. Benutzen Sie **keine** Listenfunktionale.

2. Aufgabe (9 Punkte): Listenfunktionale

Gegeben sind untenstehende Datentypen zur Verwaltung der Vorgänge im Meldeamt. Im Typ `anliegen` werden die Standardanliegen zusammengefasst. Die einzelnen Aufträge mit ihren Eigenschaften werden mit dem Datentyp `auftrag` definiert. Für jeden Auftrag werden die Art der Aktivität (`art`), der bearbeitende Schalter (`schalter`), die Wartezeit bis zum Bearbeitungsbeginn in Sekunden (`wZeit`) und die eigentliche Bearbeitungszeit in Sekunden (`bZeit`) vermerkt.

```
DATA anliegen == pass ausweis meldung bescheinigung

DATA auftrag == auftrag(art:anliegen,
                        schalter:nat,
                        wZeit:nat,
                        bZeit:nat)
```

2.1. Zeiterfassung (2 Punkte) Deklarieren und definieren Sie eine Funktion `time`, die aus einer Sequenz von Aufträgen, die Summe der Bearbeitungszeiten für alle Schalter liefert. Benutzen Sie **Listenfunktionale**. Schreiben Sie **keine** rekursiven Funktionen.

2.2. Langsamer Service (2 Punkte) Deklarieren und definieren Sie eine Funktion `slowService`, die aus einer Sequenz von Aufträgen, diejenigen Aufträge herausfiltert, deren Bearbeitungszeit über einer als Parameter zu übergebenen Zeit liegt. Benutzen Sie **Listenfunktionale**. Schreiben Sie **keine** rekursiven Funktionen.

2.3. Langsamster Schalter (5 Punkte) Deklarieren und definieren Sie eine Funktion `slowest`, die aus einer Sequenz von Aufträgen und einem als Parameter zu übergebenen Diskriminator für Anliegen (`anliegen -> bool`), denjenigen Schalter ermittelt, der für dieses Anliegen die größte Bearbeitungszeit (`bZeit`) hat. Benutzen Sie **Listenfunktionale**. Schreiben Sie **keine** rekursiven Funktionen.

3. Aufgabe (2 Punkte): Lambda-Kalkül

3.1. Substitution (2 Punkte) Substituieren Sie in dem Term $\lambda y.((x\ y)\ (\lambda x.x))$ schrittweise x durch den Term $(x\ y)$. Führen Sie, wenn nötig, α-Konversionen durch.

$(\lambda y.((x\ y)\ (\lambda x.x)))[(x\ y)/x] = \ldots$

4. Aufgabe (3 Punkte): Aufwand

4.1. Rekurrenzgleichungen (3 Punkte) Bestimmen Sie den Aufwandsterm der Opal-Funktion `toSet` in Abhängigkeit der Länge der übergebenen Liste und leiten Sie daraus die Aufwandsklasse ab. Berechnen Sie zu diesem Zweck auch den Aufwandsterm sowie die Aufwandsklasse der Funktion `in?`. Zur korrekten Lösung gehört jeweils die Angabe der verwendeten Zeile in der Tabelle sowie die Berechnung der entsprechenden Wertebelegung und die Angabe der daraus resultierenden Aufwandsklasse.

	Rekurrenzrelation		$A \in$
1	$A(n) = A(n-1) + bn^k$		$\mathcal{O}(n^{k+1})$
2	$A(n) = cA(n-1) + bn^k$	mit $c > 1$	$\mathcal{O}(c^n)$
3	$A(n) = cA(n/d) + bn^k$	mit $c > d^k$	$\mathcal{O}(n^{\log_d c})$
4	$A(n) = cA(n/d) + bn^k$	mit $c < d^k$	$\mathcal{O}(n^k)$
5	$A(n) = cA(n/d) + bn^k$	mit $c = d^k$	$\mathcal{O}(n^k \log_d n)$

```
FUN toSet: seq[nat] -> seq[nat]
DEF toSet(<>) == <>
DEF toSet(a::A) == LET b    == a in? A
                       Aset == toSet(A)
                   IN
                       IF b THEN Aset ELSE a::Aset FI

FUN in?: nat ** seq[nat] -> bool
DEF x in? <> == false
DEF x in? (a::A) == (x=a) or (x in? A)
```

5. Aufgabe (10 Punkte): Bäume

5.1. Allgemeine Fragen (2 Punkte) Welche der folgenden Aussagen sind richtig?

Ja	Nein	
○	○	Entfernt man in einem Baum mit mindestens 2 Knoten eine Kante, erhält man immer einen Wald.
○	○	Ein Heap ist ein linksvoller Binärbaum.
○	○	Zum "Reparieren" der Heap-Eigenschaft zwischen Wurzel und Nachfolgern in einem Heap benötigt man mindestens linearen Aufwand bezüglich der Anzahl der Knoten.
○	○	Die Wurzel eines beliebigen Suchbaums hat niemals Nachfolger.

5.2. Traversierung (2 Punkte) Gegeben sei der Baum `T` in Abbildung 1.

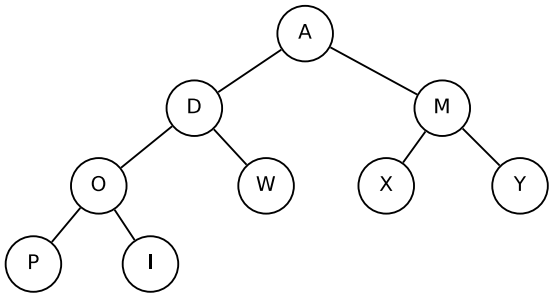


Abbildung 1: Baum T

a) Geben Sie die Folge der Knoten von Baum `T` in Preorder-Reihenfolge an.

b) Geben Sie die Folge der Knoten von Baum `T` in Inorder-Reihenfolge an.

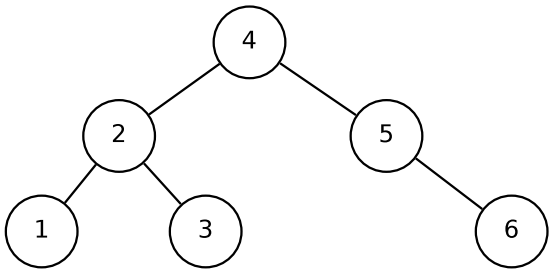
c) Geben Sie die Größe und die Höhe von Baum `T` an.

d) Intepretieren Sie den Baum `T` als Heap. Welches Array ergibt sich?

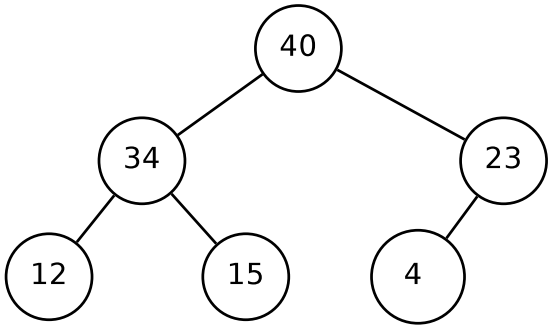
5.3. AVL-Bäume (3 Punkte) Fügen Sie in einen anfangs leeren AVL-Baum die Werte der Folge `F=10,5,15,3,7,9` nacheinander ein. Stellen Sie jeden Schritt grafisch dar und führen Sie, wenn nötig, entsprechende Rotationen durch und benennen Sie diese.

5.4. Bäume (3 Punkte) Welche der folgenden Bäume sind AVL-Bäume oder Heaps? Begründen Sie ihre Antwort kurz.

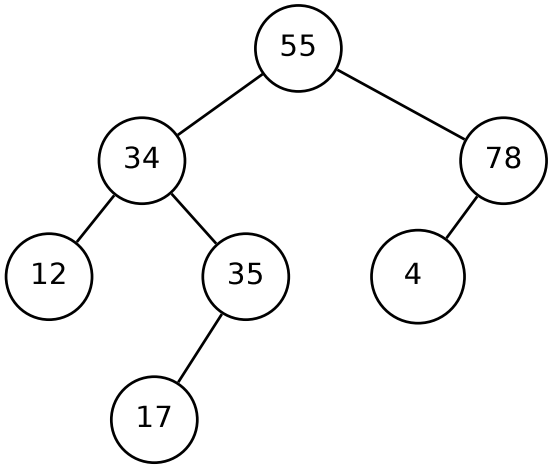
a)



b)



c)



6. Aufgabe (2 Punkte): Sortierverfahren

6.1. (1 Punkt) Abgebildet ist die Sortierung einer Sequenz von natürlichen Zahlen mit einem aus der Vorlesung bekannten Sortierverfahren. Geben Sie für den unten abgebildeten Ablauf an, mit welchem Sortierverfahren die Sortierung erfolgt ist.

```
< >   < 7 , 6 , 2 , 8 , 3 >
< 7 >     < 6 , 2 , 8 , 3 >
< 6 , 7 >     < 2 , 8 , 3 >
< 2 , 6 , 7 >     < 8 , 3 >
< 2 , 6 , 7 , 8 >     < 3 >
< 2 , 3 , 6 , 7 , 8 >   < >
```

6.2. (1 Punkt) Im Folgenden ist das aus der Vorlesung bekannte *Divide and Conquer*-Funktional gegeben:

Die Typparameter `p` und `s` dienen zur Beschreibung von Problemen (***p**roblems*) und Lösungen (***s**olutions*).

```
FUN divideAndConquer : (p->bool) -> (p->s) -> (p->seq[p]) -> (p->seq[s]->s) -> p -> s

DEF divideAndConquer(indivisible)(solve)(divide)(combine)(problem) ==
    IF indivisible(problem) THEN solve(problem)
    ELSE combine(problem)(map(d_c,divide(problem))) FI
  WHERE
    d_c == \\p.divideAndConquer(indivisible)(solve)(divide)(combine)(p)
```

Welches Sortierverfahren wird durch die folgende Funktion `sort` implementiert?

```
FUN sort : seq[nat] -> seq[nat]
DEF sort == LET
    a == \\s. #(s) < 2
    b == \\s.(filter(_ < ft(s))(rt(s))) ::
             (filter(_ = ft(s))(rt(s))) ::
             (filter(_ > ft(s))(rt(s))) :: <>
    c == \\p. \\s.ft(s) ++ (ft(p) :: ft(rt(s))) ++ ft(rt(rt(s)))
  IN divideAndConquer(a)(id)(b)(c)
```

7. Aufgabe (5 Punkte): Ein- und Ausgabe

7.1. Konkatenator (5 Punkte) Deklarieren und definieren Sie ein Kommando `konkatenator` mit folgender Funktionsweise: Zunächst wird eine Zahl von der Tastatur eingelesen. Dann soll eine entsprechende Anzahl von Zeilen von der Tastatur eingelesen werden. Abschließend wird die Konkatenation der Zeilen ausgegeben.