Klausur (Probeklausur) Informatik 1

13. Februar 2002

Jähnichen / Flik
Jucknath

Name: ....

Vorname: ....

Matr.-Nr.: ....

Bearbeitungszeit: 60 Minuten

- Dies ist eine Probeklausur, die Ihnen ein Gefühl dafür vermitteln soll, wie Sie in der echten Klausur unter Zeitdruck an Aufgaben herangehen sollen. Sie können diese Probeklausur am 13.Februar 2002 um 10 Uhr unter echten Klausurbedingungen schreiben, müßen die Klausur aber abschließend zuhause SELBST mit der von uns gelieferten Musterlösung korrigieren.
- Diese Probeklausur umfaßt nicht den vollständigen Themenbereich von Informatik 1. Bitte machen Sie sich klar, daß Sie für die echte Informatik 1 Klausur den VOLLSTÄNDIGEN STOFF DER INFORMATIK 1 VORLESUNG BEHERRSCHEN MÜSSEN und nicht nur die hier angesprochenen Themen.
- Da diese Klausur nur über einen Zeitrahmen von 60 Minuten verfügt anstatt der 120 Minuten einer echten Klausur, müssen Sie entsprechend weniger Aufgaben bearbeiten. Alle Aufgaben, die in dieser Probeklausur mit Punkten versehen wurden, sollten von Ihnen innerhalb von 60 Minuten lösbar sein.
- Da Sie diese Klausur zuhause selber korrigieren werden, haben wir für Sie einige weitere Zusatzaufgaben, passend zu der jeweiligen Aufgabenstellung hinzugefügt. BITTE BEARBEITEN SIE WÄHREND DER 60 MINUTEN PROBEKLAUSUR NUR DIE MIT PUNKTEN AUSGEZEICHNETEN AUFGABEN UND NICHT DIE ZUSATZAUFGABEN. In der echten Klausur dagegen müssen Sie ALLE Aufgaben lösen!
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen können in der echten Klausur nicht gewertet werden!
- Schreiben Sie nur in **blau** oder **schwarz**. Lösungen, die mit Bleistift geschrieben sind, können in der echten Klausur nicht gewertet werden!
- Bei der Implementation von OPAL-Funktionen brauchen *keine* Importe mit angegeben zu werden, wenn Funktionen aus dem gegebenen Teil der Bibliotheca Opalica (oder aus der Struktur `Nat`, `Real`, `Seq` oder den Strukturen für die Listenfunktionale) oder sonstigen (in den Aufgaben) gegebenen Strukturen benutzt werden!
- Erscheint Ihnen eine Aufgabe mehrdeutig, wenden Sie sich an die Betreuer.
- Tragen Sie jetzt (vor Beginn der Bearbeitungszeit) auf *allen* Blättern Ihren Namen und Ihre Matrikelnummer ein.

bitte Dokument neu übersetzen

1. Aufgabe (9 Punkte): Sequenzen

In dieser Aufgabe sollen Sie einen Algorithmus implementieren, um eine geordnete Zahlenfolge zu erzeugen, diese dann zu mischen und in eine quadratische Matrix zu überführen.
Sie dürfen nur die Funktionen `::`, `++`, `rt` und `ft` aus der Struktur `Seq` benutzen, sowie alle Funktionen der Teilaufgaben (`numbers`, `reverse`, `middle`, `split`, `mergeReverse` und `quad`)

1.1. Erstellen einer Sequenz (1 Punkte) Deklarieren und definieren Sie eine Funktion `numbers`, die eine natürliche Zahl n übergeben bekommt und eine Sequenz zurückgibt, die alle natürlichen Zahlen von 1 bis n^2 in aufsteigender Reihenfolge enthält.

```
Beispiel:
Wenn n = 3, dann ist s = <1,2,3,4,5,6,7,8,9>
```

Deklarieren und definieren Sie eine Funktion `reverse`, die eine Sequenz s von natürlichen Zahlen übergeben bekommt und eine Sequenz r zurückgibt, in der alle Elemente aus s in umgekehrter Reihenfolge stehen.

```
Beispiel:
Wenn s = <1,2,3> dann ist r = <3,2,1>
```

Lösung:

```
FUN numbers :  nat -> seq[nat]
FUN numbersHelp :  nat ** seq[nat] -> seq[nat]

DEF numbers(n) == numbersHelp(n*n, <>)
DEF numbersHelp(0, s) == s
DEF numbersHelp(x, s) == numbersHelp(x-1, x::s)

FUN reverse : seq[nat] -> seq[nat]
FUN reverseHelp : seq[nat] ** seq[nat] -> seq[nat]

DEF reverse(s) == reverseHelp(s, <>)
DEF reverseHelp(<>, s) == s
DEF reverseHelp(x::s, xs) == reverseHelp(s, x::xs)
```

1.2. Abspalten einer Teilsequenz (2 Punkte) Deklarieren und definieren Sie eine Funktion `middle`, die eine Sequenz s von natürlichen Zahlen übergeben bekommt und eine Sequenz m zurückgibt, die s ohne das erste und ohne das letzte Element enthält.

```
Beispiel:
Wenn s = <1,2,3,4> dann ist m = <2,3>
```

Deklarieren und definieren Sie eine Funktion `split`, die eine Sequenz s von natürlichen Zahlen und eine natürliche Zahl n übergeben bekommt und zwei Sequenzen s1 und s2 zurückgibt, von denen die eine alle Elemente aus s vom ersten bis zum n-ten Element enthält und die andere alle Elemente aus s vom (n+1)-ten bis zum letzten Element. Falls n grösser als die Länge der Sequenz s ist, soll s1 die Sequenz s vollständig übernehmen und für s2 eine leere Liste erzeugt werden.

```
Beispiel:
Wenn s = <4,5,6,7,8>, n = 3 dann sind s1 = <4,5,6> und s2 = <7,8>
```

Lösung:

```
FUN middle : seq[nat] -> seq[nat]
DEF middle (s) == reverse(rt(reverse(rt(s))))

FUN split :  nat ** seq[nat] -> seq[nat] ** seq[nat]

DEF split(_, <>) == (<>, <>)
DEF split(0, s)  == (<>, s)
```

```
DEF split(n, a::s) == LET
                          (s1, s2) == split(n-1, s)
                       IN
                          (a::s1, s2)
```

1.3. Mischen einer Sequenz (3 Punkte) Deklarieren und definieren Sie eine Funktion `mergeReverse`, die eine Sequenz s von natürlichen Zahlen, eine natürliche Zahl n und eine natürliche Zahl c übergeben bekommt und die Sequenz s so mischt, daß ab dem n-ten Element in s die folgenden c Elemente in ihrer Reihenfolge umgedreht werden. Diese neu gemischte Sequenz s2 von der Funktion `mergeReverse` zurückgegeben werden.

```
Beispiel:
Wenn s = <1,2,3,4,5,6,7>, n = 3, c = 4 dann ist s2 = <1,2,6,5,4,3,7>
```

Lösung:

```
FUN mergeReverse : seq[nat] ** nat ** nat -> seq[nat]
DEF mergeReverse(s, n, c) == LET
                                (s1, s2) == split(n-1, s)
                                (s3, s4) == split(c, s2)
                             IN s1 ++ reverse(s3) ++ s4
```

1.4. Erzeugung einer quadratischen Matrix (3 Punkte) Deklarieren und definieren Sie eine Funktion `quad`, die eine natürliche Zahl n und eine Sequenz s von natürlichen Zahlen der Länge n^2 übergeben bekommt und eine Matrix M mit n Spalten und n Zeilen zurückgibt.

```
Beispiel:
Wenn s = <1,2,3,4,5,6,7,8,9> dann ist M = < <1,2,3>,
                                            <4,5,6>,
                                            <7,8,9> >
```

Lösung:

```
FUN quad : nat ** seq[nat] -> seq[seq[nat]]

DEF quad(n,<>) == <>
DEF quad(n, s) == LET
                     (s1, s2) == split(n,s)
                  IN s1 :: quad(n, s2)
```

Die nun folgenden Unteraufgaben sollen Sie zuhause bearbeiten. (Ursprünglich sollte auch die Probeklausur 120 Minuten dauern. In diesem Fall wären diese Teilaufgaben zu bearbeiten gewesen.)

1.5 ZUSATZAUFGABE FÜR ZUHAUSE: Mischen durch Vertauschen Deklarieren und definieren Sie eine Funktion `mergeXchange`, die zwei Sequenzen s1 und s2 von natürlichen Zahlen übergeben bekommt und zwei Sequenzen s3 und s4 zurückgibt, wobei jeweils der mittlere Teil einer Sequenz vertauscht wurde.

```
Beispiel:
Wenn s1 = <1,2,3,4>, s2 = <5,6,7,8> dann ist s3 = <1,6,7,4> und s4 = <5,2,3,8>
```

Lösung:

```
FUN mergeXchange : seq[nat] ** seq[nat] -> seq[nat] ** seq[nat]
DEF mergeXchange(s, <>) == (s, <>)
DEF mergeXchange(<>, s) == (s, <>)
DEF mergeXchange(s1,s2) == LET
                              s3 == ft(s2) ++ middle(s1) ++ ft(reverse(s2))
                              s4 == ft(s1) ++ middle(s2) ++ ft(reverse(s1))
                           IN (s3, s4)
```

1.6 ZUSATZAUFGABE FÜR ZUHAUSE: Magische Quadrate Deklarieren und definieren Sie eine Funktion `quadGenerate`, die eine natürliche Zahl n übergeben bekommt und eine Matrix der Grösse $n \times n$ zurückgibt, die alle Zahlen von 1 bis n^2 enthält und mindestens einmal gemischt wurde.

Deklarieren und definieren Sie eine Funktion `isMagic`, die eine quadratische Matrix M übergeben bekommt und prüft, ob diese einem (einfachem) magische Quadrat entspricht. (Anmerkung: Bei einem einfachen magischen Quadrat ist die Summe jeder Zeile und jeder Spalte gleich.)

```
Beispiel:
16   1   2  15   = Summe 34
 3  14  13   4   = Summe 34
 5  12  11   6   = Summe 34
10   7   8   9   = Summe 34

 =   =   =   =
34  34  34  34
```

Zur Lösung dieser Aufgabe dürfen alle bekannten Matrizen-Funktionen (etwa aus Aufgabenblatt 4) benutzt werden.

Lösung:

```
FUN quadGenerate : nat -> seq[seq[nat]]
DEF quadGenerate(n) == LET
                           (s1, s2) == split (n/2, numbers(n))
                           (s3, s4) == mergeXchange(s1,s2)
                        IN quad(n, s3 ++ s4)

FUN sum : seq[nat] -> nat
DEF sum == reduce (+, 0)

FUN isMagic : seq[seq[nat]] -> bool
FUN testMagic : nat ** seq[seq[nat]] -> bool

DEF testMagic(n, <>) == true
DEF testMagic(n, m::M) ==
                         IF n == sum(m) THEN testMagic(n,M)
                         ELSE false
                         FI

DEF isMagic(<>) == false
DEF isMagic(M)  ==
                   LET
                       x == sum(ft(M))
                       checkHorizontal == testMagic(x,M)
                    IN
                       IF checkHorizontal THEN
                         testMagic(x, transpose(M))
                       ELSE false
                       FI
```

2. Aufgabe (9 Punkte): Datentypen, Funktionen höherer Ordnung, Bäume

Herr Gerhard S. aus Berlin, Mister George B. aus Washington und Senor Eduardo D. aus Buenos Aires haben jeweils verschiedene Währungen (Euro, Dollar oder Peso) und verschiedene Wechselkurse, sowie Steuern und Zollabgaben. Alle drei haben (sehr stark vereinfacht) die gleiche Grundstruktur bei ihren Preisen im Inland, bei Export und Import.

```
DATA preis  ==  Inland(basisPreis: real, steuern: real -> real)
                Import(basispreis: real, steuern: real -> real, zoll: real -> real)
                Export(basisPreis: real, steuern: real -> real, wechselkurs: real -> real)
```

(Ein Wort an die Wirtschafts-Ingenieure unter Ihnen: In der Realität ist dies sicherlich sehr viel komplizierter, aber wir benutzen dieses einfache Beispiel um etwas über funktionale Programmierung zu verdeutlichen. Nicht über die Weltwirtschaft.)

2.1. Induzierte Signatur (1 Punkte) Geben Sie zu dem oben definierten Datentyp `preis` die induzierte Signatur an. Wie nennt man die einzelnen Elementе?

Lösung:

```
--------------------------Sorte:
SORT preis
--------------------------Konstruktoren:
FUN Inland: real ** (real -> real) -> preis
FUN Import: real ** (real -> real) ** (real -> real) -> preis
FUN Export: real ** (real -> real) ** (real -> real) -> preis
--------------------------Selektoren:
FUN basisPreis: preis -> real
FUN steuern: preis -> real -> real
FUN zoll: preis -> real -> real
FUN wechselkurs: preis -> real -> real
--------------------------Diskriminatoren:
FUN Inland? : preis -> bool
FUN Import? : preis -> bool
FUN Export? : preis -> bool
```

2.2. Funktionen höherer Ordnung (2 Punkte) *Herr S. hat beschlossen vor der nächsten Wahl noch eine Steuererhöhung auf alle Inland-Preise zu erheben.*
Deklarieren und definieren Sie eine Funktion `steuerErhöhung`, die einen preis p und eine Funktion f: real → real bekommt und auf die Steuern bei einem Inland-Preis die Funktion f anwendet.

Lösung:

```
FUN steuerErhoehung : (real -> real) ** preis -> preis
DEF steuerErhoehung(f)(Inland(B,S)) == Inland(B,\\x.f(S(x)))
DEF steuerErhoehung(_)(P) == P
```

2.3 ZUSATZAUFGABE FÜR ZUHAUSE *Mister B. dagegen muß aufgrund aktueller Ereignisse die Zollabgaben auf alle Produkte, die in sein Land importiert werden, um einen festen Betrag (=erhöhte Aufwandsentschädigung) erhöhen.*
Deklarieren und definieren Sie eine Funktion `zollAufschlag`, die einen preis p und ein Konstante deltaZ bekommt und diese Konstante auf den Zoll bei einem Import-Preis aufaddiert.

Lösung:

```
FUN zollAufschlag : real ** preis -> preis
DEF zollAufschlag(deltaZ)(Import(B,S,Z)) == Import(B,S, \\x.Z(x) + deltaZ)
DEF zollAufschlag(_)(P) == P
```

2.4 ZUSATZAUFGABE FÜR ZUHAUSE *Senor D. musste leider feststellen, daß der alte Wechselkurs seines Landes völlig überholt ist und durch einen neuen Wechselkurs ersetzt werden muss.*
Deklarieren und definieren Sie eine Funktion `inflation`, die einen preis p und eine Funktion f: real → real bekommt und einen neuen preis zurückgibt, bei dem f die Wechselkurs-Funktion ersetzt.

Lösung:

```
FUN inflation : (real -> real) ** preis -> preis
DEF inflation(f)(Export(B,S,W)) == Export(B,S,f)
DEF inflation(_)(P) == P
```

2.5. Funktionale (3 Punkte) *Nach all den Steuerhöhungen, Zollaufschlägen und neuen Wechselkursen ist der Einzelhändler Willy K. aus Hannover völlig verwirrt und würde gerne wissen, was er an seinen Produkten im Inland noch verdient.*

Deklarieren und definieren Sie eine Funktion `verkaufspreis`, die eine Sequenz von preis bekommt und für Inlands-Preise eine Sequenz von real zurückgibt, wobei jedes Element der neuen Sequenz die Summe aus basisPreis und dem basisPreis nach Steuern ist.

```
Beispiel:
Wenn basisPreis = 5.6
und steuern(5.6) = 5.6 * 0.25
dann ist verkaufspreis = basispreis + steuern(basispreis)
                       = 5.6 + 5.6 * 0.25
                       = 7
```

Lösung:

```
FUN verkaufspreis : seq[preis] -> seq[real]
FUN verkaufspreis(s) ==
                        LET
                           min_s == filter (Inland?)(s)
                           aufschlag == \\x.basisPreis(x) + steuern(x)(basisPreis(x))
                        IN
                           map(aufschlag)(min_s)
```

2.6. Bäume (3 Punkte) *Nachdem Herr K. nun weiß, bei welchen Produkten er trotz Steuererhöhung noch etwas verdienen kann, möchte er seinen Lagerbestand überprüfen. Sein Lager verwaltet Herr K. mittels einer Baumstruktur, wobei ähnliche Produkte jeweils in Teilbäumen zusammengefasst sind. In den Blättern, die links gespeichert sind, verwaltet Herr K. die vorrätigen Produkte, in den Blättern die rechts gespeichert sind, verwaltet Herr K. die Reklamationen.*

```
Beispiel:
                Tiere
           /           \
    Katzen          Hunde
    /    \          /    \
Angora  Siam     Dackel Pudel
 /  \    / \      /  \    /  \
3    8  5   4   1    42 0    23
```

In seiner Abteilung für Haustiere hat Herr K. also noch 3 Angorakatzen, 5 Siamkatzen, 1 Dackel und keinen Pudel vorrätig, dagegen mußte er 8 Reklamationen für Angorakatzen, 4 Reklamationen für Siamkatzen, 42 Reklamationen für Dackel und 23 Reklamationen für Pudel verwalten. Was natürlich nichts über die Qualität der Dackel und Pudel von Herrn K. aussagen soll.

Herr K. benutzt also folgende Datenstruktur:

```
DATA Baum == Blatt(anzahl: nat)
             Knoten(links: Baum, rechts:Baum, beschreibung: denotation)
```

Deklarieren und definieren Sie eine Funktion `lagerbestand`, die einen Baum b bekommt und eine natürliche Zahl n zurückgibt, wobei n die Anzahl aller vorrätigen Produkte (= links gespeichertes Blatt) ist.

Lösung:

```
FUN lagerbestand : Baum -> nat
DEF lagerbestand(Knoten(Blatt(n), _, _) == n
DEF lagerbestand(Knoten(L, R, _) == lagerbestand(L) + lagerbestand(R)
```

3. Aufgabe (6 Punkte): Zahlendarstellungen

3.1. (1 Punkte) Stellen Sie 110010101101_2 zur Basis 8 und 16 dar.

Lösung:

$110010101101_2 = 6255_8 = CAD_{16}$

3.2. (1 Punkte) Wandeln Sie 469_{10} in eine vorzeichenlose Dualzahl um. Die Rechenschritte müssen erkennbar sein!

Lösung:

$$\begin{array}{rcl}
469/2 = & 234 & \mathit{Rest}\ 1 \\
234/2 = & 117 & \mathit{Rest}\ 0 \\
117/2 = & 58 & \mathit{Rest}\ 1 \\
58/2 = & 29 & \mathit{Rest}\ 0 \\
29/2 = & 14 & \mathit{Rest}\ 1 \\
14/2 = & 7 & \mathit{Rest}\ 0 \\
7/2 = & 3 & \mathit{Rest}\ 1 \\
3/2 = & 1 & \mathit{Rest}\ 1 \\
1/2 = & 0 & \mathit{Rest}\ 1 \\
469_{10} = & 111010101_2 &
\end{array}$$

3.3. (2 Punkte) Addieren Sie im Dualsystem 0110_2 und 0111_2 und geben Sie die resultierenden Bedingungsbits an. Notieren Sie die auftretenden Überträge.

$$\begin{array}{ccccc}
 & 0 & 1 & 1 & 0 \\
+ & 0 & 1 & 1 & 1 \\
\hline
 & & & & \\
\hline\hline
\end{array}$$

Lösung:

$$\begin{array}{ccccc}
 & 0 & 1 & 1 & 0 \\
+ & 0 & 1 & 1 & 1 \\
 & \bullet & \bullet & & \\
\hline
 & 1 & 1 & 0 & 1 \\
\hline\hline
\end{array}$$ c=0, z=0, n=1, v=1

3.4. (1 Punkte) Subtrahieren Sie im Dualsystem 1010_2 und 0111_2. Notieren Sie die auftretenden Überträge.

$$\begin{array}{ccccc}
 & 1 & 0 & 1 & 0 \\
- & 0 & 1 & 1 & 1 \\
\hline
 & & & & \\
\hline\hline
\end{array}$$

Lösung:

$$\begin{array}{ccccc}
 & 1 & 0 & 1 & 0 \\
- & 0 & 1 & 1 & 1 \\
 & \bullet & \bullet & \bullet & \\
\hline
 & 0 & 0 & 1 & 1 \\
\hline\hline
\end{array}$$

3.5. (1 Punkte) Wandeln Sie $+0.00101001_2$ in das aus der Vorlesung bekannte Gleitpunktformat (1-Bit-Vorzeichen, 4-Bit-Exponent, 11-Bit-Mantisse, Bias = 7) um.

Lösung:

Normalisieren:	$0.00101001_2 = 1.01001_2 \cdot 2^{-3}$
Exponent:	$-3 + 7 = 4 = 0100_2$
Ergebnis:	*0 0100 01001000000*

4. Aufgabe (2 Punkte): Boolesche Algebra

4.1. **(2 Punkte)** Vergleichen Sie die Ausdrücke $\overline{a}+\overline{b}+\overline{c}$ und $ab \rightarrow \overline{c}$ mit Hilfe der folgenden Tabelle.

a b c	$\overline{a}+\overline{b}$	$\overline{a}+\overline{b}+\overline{c}$	ab	$ab \rightarrow \overline{c}$

Lösung:

a	b	c	$\overline{a}+\overline{b}$	$\overline{a}+\overline{b}+\overline{c}$	ab	$ab \rightarrow \overline{c}$
0	0	0	1	1	0	1
0	0	1	1	1	0	1
0	1	0	1	1	0	1
0	1	1	1	1	0	1
1	0	0	1	1	0	1
1	0	1	1	1	0	1
1	1	0	0	1	1	1
1	1	1	0	0	1	0

$$\overline{a}+\overline{b}+\overline{c} = ab \rightarrow \overline{c}$$