Fachbereich Informatik

Klausur Informatik 1
12. Oktober 2000

Mahr / Lemke
Weisweber / Tschirley

Name: ....

Vorname: ....

Matr.-Nr.: ....

Bearbeitungszeit: 120 Minuten

- Benutzen Sie für die Lösung der Aufgaben nur das mit diesem Deckblatt ausgeteilte Papier. **Lösungen, die auf anderem Papier gerschrieben werden, können nicht gewertet werden!**
- Schreiben Sie Ihre Lösungen auf das Aufgabenblatt der jeweiligen Aufgabe. Verwenden Sie auch die Rückseite. **Schreiben Sie keine Lösungen einer Aufgabe auf ein Blatt, das nicht zu dieser Aufgabe gehört!**
- Schreiben Sie deutlich! Doppelte, unleserliche oder mehrdeutige Lösungen können nicht gewertet werden.
- Schreiben Sie nur in **blau** oder **schwarz**. Schreiben Sie nicht mit Bleistift!
- 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`) oder sonstigen (in den Aufgaben) gegebenen Strukturen benutzt werden!
- Die OPAL-Implementierungen müssen nicht dokumentiert 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.

Aufgabe	Punkte	erreicht
1	6	
2	6	
3	7	
4	7	
5	7	
6	6	
7	10	

1. Aufgabe (6 Punkte): Softwareentwicklungszyklus

Beschreiben Sie den grundlegenden Softwareentwicklungszyklus

a) unter Verwendung eines Compilers und

b) unter Verwendung eines Interpretierers

in Form eines Ablaufdiagramms oder in Worten. Geben Sie dabei die verwendeten Werkzeuge und deren Ein- und Ausgabedaten an.

Lösung:

a) Verwendung eines Compilers

a) *Mit einem **Texteditor** wird ein **Programm** erstellt.*

b) *Das Programm wird mit einem **Compiler** compiliert, der daraus **Maschinencode** erstellt. Im Falle von **Syntaxfehlern** werden entsprechende **Fehlermeldungen** ausgegeben.*

c) *Der erzeugte Maschinencode wird in einer Laufzeitumgebung (dieses Wort ist in der Vorlesung nicht gefallen) **ausgeführt**, die **Benutzereingaben** liest und daraufhin eine **Ausgabe** erzeugt. Im Falle von **Laufzeitfehlern** werden entsprechende **Fehlermeldungen** erzeugt.*

d) *Falls Syntax- oder Laufzeitfehler vorkommen, muß das Programm **neu editiert (korrigiert)** werden und der Entwicklungszyklus beginnt von vorne.*

e) *Wenn das Programm fehlerfrei abläuft, muß es ausführlich **getestet** werden (Normal- und Grenzfälle, d.h. Extrem- und Ausnahmefälle).*

f) *Parallel zu allen Phasen ist das Programm zu **dokumentieren** (Kommentare im Programm und Handbücher).*

b) Verwendung eines Interpretierers

Wie unter Verwendung eines Compilers mit einer Ausnahme:

- *Die Compilierungs- und Ausführungsphase werden zu einer Phase **vereinigt** und es gilt sinngemäß das oben unter b. und c. Gesagte, d.h. im Falle von Syntax- oder Laufzeitfehlern werden entsprechende **Fehlermeldungen** ausgegeben.*

2. Aufgabe (6 Punkte): Abstrakte Datentypen

Definieren Sie einen abstrakten Datentypen `grad` mit passendem Signatur- und Implementierungsteil, der Berechnungen und Konvertierungen in die drei Temperatureinheiten Grad Celsius (`inC`), Kelvin (`inK`) und Grad Fahrenheit (`inF`) ermöglicht. Dabei ist 0 Grad Kelvin der absolute Nullpunkt und entspricht -273,16 Grad Celsius. 32 Grad Fahrenheit ist der Gefrierpunkt, d.h. entspricht 0 Grad Celsius und 212 Grad Fahrenheit der Siedepunkt, d.h. 100 Grad Celsius. Die Umrechnungsformel lautet also $c = (f - 32)/1.8$, wenn c die Temperatur in Grad Celsius und f die in Grad Fahrenheit ist. Jede Temperatureinheit soll darstellbar sein (z.B. `25 C`, `28 K` oder `31 F`) und in die jeweils anderen Einheiten konvertierbar sein. Als Operationen auf den Temperatureinheiten sollen `+`, `/` und `durchschnitt` realisiert werden, wobei `+` zwei Temperatureinheiten als Argumente nimmt und `/` eine Temperatureinheit als erstes Argument und einen **nat**-Wert als zweites. Die Funktion `durchschnitt` berechnet die Durschnittstemperatur aus einer vorzugebenden Liste von Temperaturen.

Hinweise:

- Als Datentyp `grad` soll ein Produktyp definiert werden, der die Temperatur in Grad Celsius angibt und **kein Summentyp**, der aus allen drei Temperatureinheiten besteht. Grad Kelvin und Grad Fahrenheit sind dann als einstellige Funktionen `K` bzw. `F` von `real` nach `grad` zu realisieren.
- Die Funktion `#` aus der Struktur `Seq[alpha]` mit der Signatur `FUN # : seq[alpha] -> nat` kann zur Bestimmung der Länge einer Liste verwendet werden.
- Die Funktion `asReal` aus der Struktur `NatConv` mit der Signatur `FUN asReal : nat -> real` kann zur Konvertierung von `nat` nach `real` verwendet werden.

Lösung:

```
IMPLEMENTATION Grad

IMPORT Real COMPLETELY
IMPORT NatConv ONLY asReal
IMPORT Seq[grad] COMPLETELY
IMPORT SeqReduce[grad,grad] COMPLETELY

DATA grad == C(wert : real)

DEF x K == C(x - !("273.16"))
DEF x F == C((x-32)/ !("1.8"))
DEF temp inC == wert(temp)
DEF temp inK == (temp inC)+ !("273.16")
DEF temp inF == (temp inC)* !("1.8")+32
DEF g1 + g2 == C(wert(g1)+wert(g2)) -- fuer durchschnitt
DEF g / r == C(wert(g)/ asReal(r))  -- fuer durchschnitt
DEF durchschnitt == \\ liste. reduce(+,C(0))(liste)/ #(liste)
```

3. Aufgabe (7 Punkte): Datentypen und Sortieren

3.1. Definition (2 Punkte) Definieren Sie einen neuen Datentypen `persdaten`, der insgesamt Informationen über den Vornamen (`denotation`), Nachnamen (`denotation`), Straße (`denotation`), Hausnummer (`nat`), Postleitzahl (`nat`) und den Wohnort (`denotation`) einer Person aufnehmen kann. Dabei sollen Vor- und Nachname sowie die Adressdaten (Straße, Hausnummer, Postleitzahl und Wohnort) jeweils zu eigenen Datentypen `name` bzw. `adresse` zusammengefaßt werden.

Lösung:

```
DATA persdaten == persdaten(nam : name, adr : adresse)
DATA name == name(vor : denotation, nach : denotation)
DATA adresse ==
        adresse(
                strasse : denotation,
                hausnummer : nat,
                postleitzahl : nat,
                wohnort : denotation)
```

3.2. Induzierte Signatur (3 Punkte) Geben Sie die vollständigen Signaturen an, die durch die Deklaration der Datentypen `persdaten`, `name` und `adresse` induziert werden und geben Sie die Kategorien der einzelnen Funktionen an (Konstruktor, Selektor, Diskriminator).

Lösung:

```
SORT persdaten
SORT name
SORT adresse
-- Konstruktorfunktion
FUN persdaten : name ** adresse -> persdaten
FUN name : denotation ** denotation -> name
FUN adresse : denotation ** nat ** nat ** denotation -> adresse
-- Selektorfunktionen
FUN nam : persdaten -> name
FUN adr : persdaten -> adresse

FUN vor : name -> denotation
FUN nach : name -> denotation

FUN strasse : adresse -> denotation
FUN hausnummer : adresse -> nat
FUN postleitzahl : adresse -> nat
FUN wohnort : adresse -> denotation
-- Diskriminatoren (werden von Opal erzeugt, sind hier aber nutzlos)
FUN persdaten? : persdaten -> bool
FUN name? : name -> bool
FUN adresse? : adresse -> bool
```

3.3. Sortieren von Adressen (2 Punkte) Schreiben Sie eine Funktion, die eine Liste von Personen des Typs `persdaten` nach den Nachnamen sortieren kann. Als Sortierverfahren soll der Quick-Sort verwendet werden:

a) Nimm das erste Element e einer Liste.

b) Teile die Restliste r in zwei Listen k und g auf, wobei k sämtliche Elemente enthält, die kleiner oder gleich den Element e sind, und g alle Elemente, die größer sind als e.

c) Sortiere die beiden Listen k und g (natürlich wieder mit Quick-Sort).

d) Die sortierte Gesamtliste enthält man durch Konkatenation der sortierten Liste k, des Elements e und der sortierten Liste g in der angegebenen Reihenfolge.

Lösung:

```
-- Vergleiche

FUN vergleiche : persdaten ** (denotation ** denotation -> bool)
        ** persdaten -> bool

DEF vergleiche == \\p1,vergleich,p2.
        nach(nam(p1)) vergleich nach(nam(p2))

-- Sortieren von Personen mit Quicksort

FUN sortiere : seq[persdaten] -> seq[persdaten]

DEF sortiere(<>) == <>
DEF sortiere(el::rest) ==
        LET     kleine == filter(vergleiche(_,<,el))(rest)
                mittlere == el :: filter(vergleiche(_,=,el))(rest)
                grosse == filter(vergleiche(_,>,el))(rest)
        IN      sortiere(kleine)++mittlere++sortiere(grosse)
```

4. Aufgabe (7 Punkte): Rekursion und Termination

4.1. (3 Punkte) In funktionalen Sprachen wie Opal ist die Rekursion die wichtigste Programmiertechnik. Dabei ist besonders auf die Termination zu achten. Bitte geben Sie an, auf was Sie achten müssen, damit ein (funktionales) Programm terminiert.

Lösung:

a) ***Reduktion*** *(z.B. Subtraktion) eines Eingabearguments des rekursiven Aufrufs, bis eine* ***untere Grenze*** *erreicht ist, oder*
b) ***Expansion*** *(z.B. Addition) eines Eingabearguments, bis eine* ***obere Grenze*** *erreicht ist.*
c) *Ein rekursive Funktionen wird immer durch eine* ***Fallunterscheidung*** *realisiert.*
d) *Es muß ein Fall (Ausdruck) ohne rekursiven Aufruf vorkommen (**Rekursionsabschluß**)*

4.2. (4 Punkte) Sehen Sie sich die folgenden beiden Funktionen an:

```
FUN .. : nat ** nat -> seq[nat]
DEF n .. m ==
        IF n > m THEN <>
        ELSE n :: ((n+1) .. m)
        FI

FUN funktion : nat ** seq[nat] -> bool
DEF funktion(_,<>) == false
DEF funktion(el1,el2::rest) ==
        IF el1 = el2 THEN true
        ELSE funktion(el1,rest)
        FI
```

a) Beschreiben Sie, was die Funktionen `..` und `funktion` machen (Semantik).
b) Begründen Sie, ob und, wenn ja, warum die Funktionen terminieren. Der Nachweis kann in eigenen Worten erfolgen. Ein formaler Beweis ist möglich, aber nicht notwendig.

Lösung:

Funktion ..

Semantik: Erzeugt eine Liste mit $(m - n + 1)$ Elementen. Die Elemente sind dabei die Zahlen n bis m (inklusive).
Termination: Fallunterscheidung

a) *Fall $n > m$ (Rekursionsabschluß): Termination offensichtlich.*
b) *Fall $n <= m$ (Rekursionsschritt): n wird um 1 erhöht und nähert sich damit immer weiter der oberen Grenze m. Sobald die obere Grenze m überschritten ist, tritt der a. Fall ein und die Funktion terminiert.*

Funktion `funktion`

Semantik: Überprüft, ob ein Element in einer Liste vorkommt.
Termination: Fallunterscheidung

a) *Fall die Liste (das zweite Argument) ist leer (Rekursionsabschluß): Termination offensichtlich*
b) *Fall die Liste (das zweite Argument) enthält mindestens ein Element:*
   a. *Fall `el1 = el2` (Rekursionsabschluß): Termination offensichtlich*
   b. *Fall `el1` $\neq$ `el2` (Rekursionsschritt): Es wird ein Element in der Liste weitergerückt, d.h. das erste Element wird weggelassen, und nähert sich damit immer weiter dem Ende der Liste (a. Fall) oder findet das gesuchte Element (ba. Fall).*

5. Aufgabe (7 Punkte): Funktionen höherer Ordnung

Deklarieren und definieren Sie die folgenden drei Funktionen höherer Ordnung:

5.1. Folgen (2 Punkte) eine Funktion `folge`, die aus einer Funktion f mit Signatur `nat -> nat` eine Folge von $f(0)$ bis $f(n)$ erzeugt. Die Folge soll so lang sein, bis $f(n)$ eine anzugegebende Schranke überschritten hat. Die Funktion soll aufgerufen werden können als `folge(f)(schranke)`.

Lösung:

```
FUN folge : (nat -> nat) -> nat -> seq[nat]
DEF folge(f)(schranke) == folge(f,schranke,0)

FUN folge : (nat -> nat) ** nat ** nat -> seq[nat]
DEF folge(f,schranke,n) ==
        LET     wert == f(n)
        IN      IF wert > schranke THEN <>
                ELSE wert::folge(f,schranke,n+1)
                FI
```

5.2. Verschiebung einer mathematischen Funktion (2 Punkte) eine Funktion `verschieben`, die eine Funktion f mit Signatur `real -> real` in x- und y-Richtung verschiebt. Die Funktion soll aufgerufen werden können als `verschieben(dx,dy)(f)(x)`, wobei `dx` und `dy` die Verschiebung angeben.

Lösung:

```
FUN verschieben : (real ** real) -> (real -> real) -> (real -> real)
DEF verschieben(dx,dy)(f)(x) == f(x-dx)+dy
```

5.3. Listenfunktionale (3 Punkte) eine Funktion `nachfahren`, die für eine Person der Sorte `person` sämtliche Nachfahren (Kinder und Kindeskinder) berechnet. Die Person selbst gehört mit zu dieser Menge. Benutzen Sie zur Lösung Listenfunktionale. `person` sei dabei definiert als:

```
DATA person == person(
        name : denotation,
        kinder : seq[person])
```

Lösung:

```
FUN nachfahren : person -> seq[denotation]
DEF nachfahren(pers) ==
        reduce(++,name(pers):: <>)(map(nachfahren)(kinder(pers)))

-- alternativ: name(pers)::reduce(++,<>)(map(nachfahren)(kinder(pers)))
```

6. Aufgabe (6 Punkte): Boolesche Algebra

6.1. (3 Punkte) Beweisen oder widerlegen Sie folgende Gleichung durch Anwendung der bekannten Rechenregeln (nicht durch eine Wertetabelle). Sorgen Sie dafür, daß die Umformungen nachvollziehbar sind!

$$(a \cdot b) + (\overline{a} \cdot \overline{b}) \quad = \quad \overline{(\overline{a} \cdot b) + (a \cdot \overline{b})}$$

Lösung:

$$\begin{aligned}
(a \cdot b) + (\overline{a} \cdot \overline{b}) &= \overline{(\overline{a} \cdot b) + (a \cdot \overline{b})} \\
&= \overline{(\overline{a} \cdot b)} \cdot \overline{(a \cdot \overline{b})} \\
&= (a + \overline{b}) \cdot (\overline{a} + b) \\
&= (a \cdot \overline{a}) + (a \cdot b) + (\overline{b} \cdot \overline{a}) + (\overline{b} \cdot b) \\
&= (a \cdot b) + (\overline{a} \cdot \overline{b})
\end{aligned}$$

6.2. (3 Punkte) Formen Sie beide Seiten der Gleichung so weit wie möglich um.

$$(a \cdot b \cdot c) + \overline{\overline{(\overline{a} \cdot b)} \cdot \overline{(\overline{a} \cdot \overline{b})}} + a \quad = \quad \overline{\overline{c} + \overline{(a \cdot b \cdot c)} + (c + c \cdot a)}$$

Lösung:
Wir betrachten beide Seiten der Gleichung getrennt:

$$\begin{aligned}
(a \cdot b \cdot c) + \overline{\overline{(\overline{a} \cdot b)} \cdot \overline{(\overline{a} \cdot \overline{b})}} + a &= a \cdot (1 + b \cdot c) + \overline{\overline{(\overline{a} \cdot b)} \cdot \overline{(\overline{a} \cdot \overline{b})}} \\
&= a + \overline{\overline{(\overline{a} \cdot b)} \cdot \overline{(\overline{a} \cdot \overline{b})}} \\
&= a + (\overline{a} \cdot b) + (\overline{a} \cdot \overline{b}) \\
&= a + \overline{a} \cdot (b + \overline{b}) \\
&= a + \overline{a} \\
&= 1
\end{aligned}$$

$$\begin{aligned}
\overline{\overline{c} + \overline{(a \cdot b \cdot c)} + (c + c \cdot a)} &= c \cdot (a \cdot b \cdot c) \cdot \overline{(c + c \cdot a)} \\
&= c \cdot (a \cdot b \cdot c) \cdot \overline{c} \cdot \overline{(c \cdot a)} \\
&= c \cdot \overline{c} \cdot (a \cdot b \cdot c) \cdot \overline{(c \cdot a)} \\
&= 0 \cdot (a \cdot b \cdot c) \cdot \overline{(c \cdot a)} \\
&= 0
\end{aligned}$$

Somit ist die Gleichung umformbar zu

$$1 = 0$$

was offensichtlich falsch ist.

7. Aufgabe (10 Punkte): Zahlendarstellungen

Kommentieren Sie Ihren Lösungsweg – Lösungen, bei denen der Lösungsweg nicht erkennbar ist, können nicht gewertet werden!

7.1. (2 Punkte) Berechnen Sie die Lösung von $243_{(5)} + 122_{(3)}$ und geben Sie das Ergebnis im Binärsystem (Basis 2) an.

Lösung:
Beide Zahlen werden in das Dezimalsystem konvertiert und addiert.

$$\begin{aligned} 243_{(5)} &= 2 \cdot 5^2 + 4 \cdot 5^1 + 3 \cdot 5^0 \\ &= 2 \cdot 25 + 4 \cdot 5 + 3 \\ &= 50 + 20 + 3 \\ &= 73 \end{aligned}$$

$$\begin{aligned} 122_{(3)} &= 1 \cdot 3^2 + 2 \cdot 3^1 + 2 \cdot 3^0 \\ &= 1 \cdot 9 + 2 \cdot 3 + 2 \\ &= 9 + 6 + 2 \\ &= 17 \end{aligned}$$

$$73 + 17 = 90$$

Nun wird das Ergebnis in das Binärsystem konvertiert:

90	div 2 =	45	Rest	0
45	div 2 =	22	Rest	1
22	div 2 =	11	Rest	0
11	div 2 =	5	Rest	1
5	div 2 =	2	Rest	1
2	div 2 =	1	Rest	0
1	div 2 =	0	Rest	1

*Das Ergebnis der Rechnung ist **0101 1010**.*

7.2. (4 Punkte) Geben Sie die beiden Dezimalzahlen $a = -77,9375$ und $b = 9,0625$ als binäre Festkommazahlen mit 8 Vor- und 4 Nachkommastellen im Zweierkomplement an.

Lösung:
Wir berechnen zunächst die Binärcodierung von 77,9375. Der Vorkommaanteil lautet binär codiert 0100 1101 (Schema oben). Der Nachkommaanteil wird separat berechnet:

0,9375	· 2 =	1,875
0,875	· 2 =	1,75
0.75	· 2 =	1,5
0,5	· 2 =	1,0

*Die 77,9375 entspricht also dem Bitmuster 0100 1101,1111, das Einerkomplement lautet 1011 0010,0000, das Zweierkopmlement der negativen Zahl lautet somit **1011 0010,0001**.*

*Die 9,0625 wird analog konvertiert und ergibt **0000 1001,0001**.*

Berechnen Sie dann *binär*

- $a + b$ und

  Lösung:
  Die Berechnung von 1011 0010,0001 + 0000 1001,0001 lautet

  ```
    0000 1001,0001
  + 1011 0010,0001
  ----------------
    1011 1011,0010
  ```

  *damit ist das Ergebnis **1011 1011,0010**.*
  *Zur Konvertierung in das Dezimalsystem wird erneut das Zweierkomplement gebildet: 0100 0100,1110 was analog rückkonvertiert wird. Es ergibt sich die **-68,875** als Ergebnis.*

- $a \cdot b$

und geben Sie das Ergebnis mit 8 Vor- und 4 Nachkommastellen an. Konvertieren Sie das Ergebnis jeweils in das Dezimalsystem. Rechnen Sie ausschließlich mit vorzeichenbehafteten Zahlen! Interpretieren Sie ihr Ergebnis.

Lösung:

Zur korrekten Multiplikation erweitern wir vor der Rechnung beide Vorkommaanteile auf eine Breite von 16 Bit: -77,9375 = 1111 1111 1011 0010,0001; 9,0625 = 0000 0000 0000 1001,0001.

Die Multiplikation lautet dann:

```
1111 1111 1011 0010,0001 * 0000 0000 0000 1001,0001
---------------------------------------------------
                         1111 1111 1011 0010 0001
                    1111 1111 1011 0010 0001
              + 111 1111 1101 1001 0000 1
       Übertrag:           32 111   1
             -----------------------------------
                         .. 1 0011 1101,1011 0001
```

Nun schneiden wir aus dem Ergebnis den darstellbaren Bereich heraus und erhalten ***0011 1101,1011****. Wir stellen fest, daß die Zahl nicht wie erwartet negativ ist, was nur dadurch erklärbar ist, daß ein Überlauf auftrat (was bestätigt wird, wenn man $77 \cdot 9 = 693$ überschlägt, was nicht mehr durch 8 Bit darstellbar ist, da $2^8 = 256$). Die Rückkonvertierung erfolt analog zu den vorigen Aufgaben und liefert dezimal* ***61,6875****.*

7.3. (4 Punkte) Gegeben ist die mit 16 Bit gespeicherte Gleitkommazahl 1010 0110 1101 1000, die nach dem in der LV vorgestellten, an dem Gleitkommazahlenformat der IEEE orientierten Schema codiert ist. Berechnen Sie den dargestellten Bruch.

Hinweis: Das Format sieht 1 Vorzeichenbit, 4 Bit für den mit dem Bias 7 versehenen Exponenten und 11 Bit für den Nachkommaanteil der Mantisse in dieser Folge vor.

Lösung:

Wir teilen die Zahl zunächst in die drei Felder auf:

1	*010 0*	*110 1101 1000*
s	*e*	*f*

Nun berechnen wir den Nachkommaanteil der Mantisse. Da der Exponent nicht gleich Null ist, liegt die Zahl in der normalisierten Darstellung vor, was bedeutet, daß der Vorkommaanteil der Mantisse gleich 1 ist. Die gesamte Mantisse lautet demnach binär 1,1101 1011 000.

Der Exponent $0100_{(2)}$ lautet dezimal 4, abzüglich des Bias von 7 lautet der Exponent also -3. Damit ergibt sich die Zahl 0,0011 1011 0110 00, die nun in das Deximalsystem konvertiert werden muß. Der Vorkommaanteil lautet 0, der Nachkommaanteil berechnet sich analog zu Aufgabenteil 1:

$$\begin{aligned}
0011\ 1011\ 0110\ 00 &= 2^{-3} + 2^{-4} + 2^{-5} + 2^{-7} + 2^{-8} + 2^{-10} + 2^{-11} \\
&= \frac{1}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{128} + \frac{1}{256} + \frac{1}{1024} + \frac{1}{2048} \\
&= \frac{256}{2048} + \frac{128}{2048} + \frac{64}{2048} + \frac{16}{2048} + \frac{8}{2048} + \frac{2}{2048} + \frac{1}{2048} \\
&= \frac{475}{2048} \\
&= 0,23193359375
\end{aligned}$$

Bei Berücksichtigung des Vorzeichen-Bits erhält man die dargestellte Zahl ***-0,231 933 593 75****.*