Otto-von-Guericke-Universität Magdeburg
Fakultät für Informatik
Institut für Technische und Betriebliche Informationssysteme
Prof. Dr. R. Dumke

Magdeburg, 18.07.08

Prüfung

Programmierparadigmen

Name:	Matrikelnummer:
Vorname:	Studiengang:
Blattanzahl:	Unterschrift:

Aufg. 1	Aufg. 2	Aufg. 3	Aufg. 4	Aufg. 5	Aufg. 6	Aufg. 7	Summe
(von 3)	(von 5)	(von 3)	(von 6)	(von 6)	(von 8)	(von 9)	(von 40)

Allgemeine Hinweise

- Schreiben Sie auf jedes Blatt Ihren Namen, Ihre Matrikelnummer und die Seitennummer.
- Die Programme sind gut zu kommentieren!
- Beginn und Ende einer Aufgabenlösung sind durch einen waagerechten Strich deutlich zu kennzeichnen.
- Ungültige Lösungen sind durchzustreichen.
- Zugelassene Hilfsmittel: ausschließlich Schreibmaterialien
- Die Klausur besteht aus 7 Aufgaben, die Bearbeitungszeit beträgt 120 Minuten.

Aufgabe 1 (3 Punkte)

Ordnen Sie folgende Programmiersprachen den Programmierparadigmen *imperativ*, *objektorientiert* und *deklarativ* zu: Java, Fortran, Haskell, C, Prolog und Pascal.

Aufgabe 2 (5 Punkte)

Gegeben sei der folgende Java- und AspectJ-Quellcodeausschnitt.

```
1: public class KlasseA {
2:    public KlasseA()
3:       {System.out.println("Neues Objekt!");}
4:    public void beispiel(int number, String name)
5:       {System.out.println("Innerhalb von beispiel (int, String)");
6:       System.out.println("beispiel-Parameter: "+number+" und "+name);}
7:    public static void main(String[] args)
8:          { . . . }  }
9: public aspect AspektDef{
10:    pointcut BerechneAspekt():
11:         execution(void KlasseA.beispiel(int, String));
12:         before(): BerechneAspekt()
13:               {System.out.println("Aspekt01");}
14:    pointcut InitAspekt():
15:         initialization(public KlasseA.new());
16:         before(): InitAspekt()
17:               {System.out.println("Aspekt02"); }
18:    pointcut AendereAspekt(int z, String n):
19:         call(void KlasseA.beispiel(int, String)) && args(z,n);
20:         after(int z, String n): AendereAspekt(z,n)
21:          {System.out.println("Aspekt03");}  }
```

Welche Zeilen gehören zu einem *Objektinitialisierungs-Joinpoint*, *Methoden-Joinpoint*, *before-Advice*, *Methodenaufruf-Pointcut* und *Methodenabarbeitungs-Pointcut*?

Aufgabe 3 (3 Punkte)

removeOdd soll aus den Elementen einer gegebenen Liste (bestehend aus ganzen Zahlen) eine neue Liste erzeugen, die nur noch die geraden Elemente der Ursprungsliste enthält.
Beispiel: *removeOdd [1,2,3,4]* ergibt *[2,4]*.

Wie sieht die Typdeklaration aus?

Definieren Sie die Funktionalität von *removeOdd* auf die folgenden Arten in *Haskell:*
- direkt als *rekursive* Funktion (removeOdd1),
- mit Hilfe einer geeigneten *Listenkomprehension* (removeOdd2) und
- mit Hilfe von *filter* und einem geeigneten Prädikat (removeOdd3).

Aufgabe 4 (6 Punkte)

a) Definieren Sie in Haskell mit Rekursion eine Funktion *applyAll*, die eine Liste von Funktionen $[f_1, f_2, .. , f_n]$ und ein Element *x* als Argument nimmt und als Wert $f_1\ (f_2\ (\ldots(f_n\ x)\ \ldots))$ liefert. Was ist der Typ von *applyAll*?
Beispiel: *aplyAll [(+ 3), (*6), (\x -> x-7)] 9 ergibt (((9-7)*6)+3) = 15.*

b) Gesucht ist eine Funktion *composeFunc*, die eine Liste von Funktionen $[f_1, f_2, .. , f_n]$ durch Komposition zu einer einzigen Funktion $f_1 \bullet (f_2 \bullet (\ldots(f_{n-1} \bullet f_n) \ldots))$ zusammenfügt.
- Was ist der Type von *composeFunc*? Begründen Sie kurz.
- Definieren Sie *composeFunc direkt mit Rekursion.*
  Hinweis: *composeFunc* soll *id* liefern, wenn es auf eine leere Liste angewendet wird.
- Definieren Sie *composeFunc* mit Hilfe von *foldr*.

c) Benutzen Sie *composeFunc* für eine alternative Definition von *applyAll.*

Aufgabe 5 (6 Punkte)

Gegeben sei der abstrakte Datentyp *Queue*:
data Queue a = Qu [a]
deriving (Show)

emptyQu	erzeugt eine leere *Queue*
isEmpty	liefert *True*, wenn *Queue* leer ist, sonst *False*
enqueue	fügt ein Element hinten an die *Queue* an
dequeue	trägt das vorderste Element des der *Queue* ab und liefert die veränderte *Queue* zurück
front	liefert das erste Element der *Queue*, ohne sie zu verändern

Geben Sie für die Methoden des obigen ADT *Queue* Implementationen in *Haskell* einschließlich der dazugehörigen Typdeklarationen an.

Aufgabe 6 (8 Punkte)

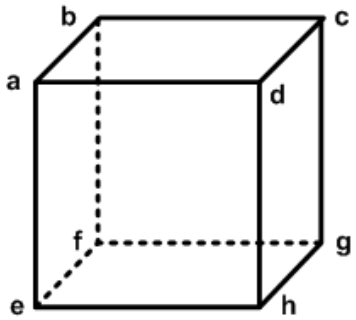
Gegeben seien die folgenden Fakten in Prolog:

```
mutter(X,Y).          % X ist Mutter von Y
vater(X,Y).           % X ist Vater von Y
weiblich(X).          % X ist weiblich
maennlich(X).         % X ist maennlich
verheiratet(X,Y).     % X ist mit Y verheiratet
```

Schreiben Sie Regeln zur Bestimmung der näheren Verwandtschaft für Oma, Sohn, Schwester, Tante, Neffe und Enkel.

Aufgabe 7 (9 Punkte)

Bestimmen Sie in einem Würfel kreisfreie Wege zwischen zwei Endpunkten und deren Länge
Hinweis: Arbeiten Sie mit einer Hilfsliste, einem Hilfszähler und dem *member*-Prädikat.



Wir wünschen Ihnen viel Erfolg!