Vordiplom Informatik für E-Techniker

SS 1999

Beispiel-Lösung

Name ______________________________

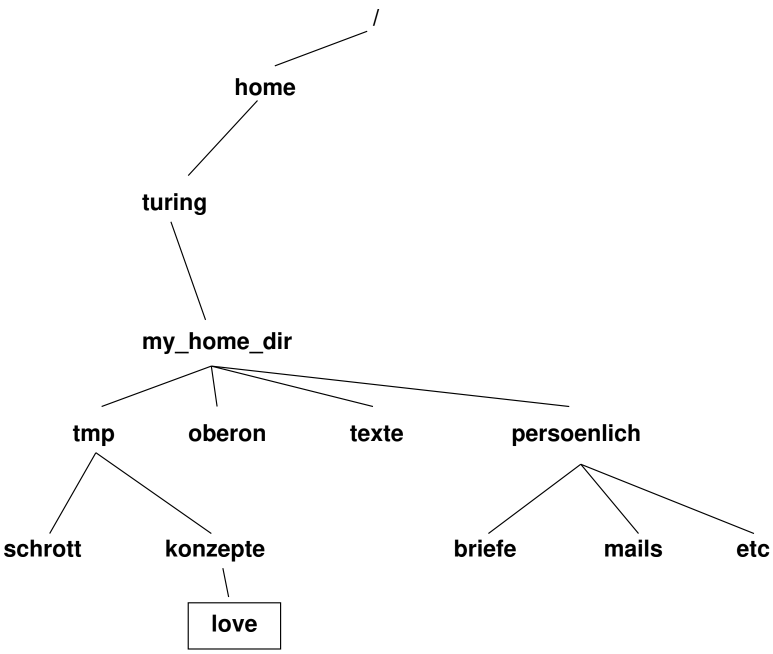
Vorname ______________________________

Matrikel-Nr. ______________________________

Aufgabe		Punkte	Bewertung
1	(Dateisystem)	6	
	a)	2	
	b)	1	
	c)	3	
2	(Unix-Programme)	10	
	a)	4	
	b) - d)	je 2	
3	(reguläre Ausdrücke)	12	
	a)	5	
	b)	7	
4	(binäre Bäume)	12	
5	(EBNF)	12	
6	(Datenstrukturen)	12	
	a) - c)	je 4	
7	(Listen)	12	
8	(Typerweiterung)	12	
	a)	4	
	b) - e)	je 2	
9	(Rekursion)	12	
Summe		**100**	

Aufgabe 1 (6 Punkte)

Gegeben sei folgender Ausschnitt aus einem UNIX-Dateisystem.



Nach dem Anmelden stehen Sie im Katalog *my_home_dir*.

(*a*) Sie wollen ohne den Katalog zu wechseln (also weiterhin in *my_home_dir*) die Datei *love* in den Katalog *briefe* verschieben. Geben Sie ein dazu notwendiges Kommando an!

(*b*) Geben Sie an, wie Sie mit einem Kommando in den Katalog *briefe* wechseln (ausgehend wieder von *my_home_dir*).

(*c*) Ihr Arbeitskatalog ist jetzt der Katalog *briefe*. Geben Sie in einer Kommandozeile an, wie Sie die Zahl aller Einträge in diesem Katalog ermitteln können.
Hinweis: `wc -l` kann prima Zeilen zählen.

(*a*) Es genügt, das Ziel-Verzeichnis anzugeben:

```
mv tmp/konzepte/love persoenlich/briefe
```

Wer es ausführlicher mag, kann auch den Dateinamen bei der Zielspezifikation mit angeben:

```
mv tmp/konzepte/love persoenlich/briefe/love
```

(*b*)

```
cd persoenlich/briefe
```

(*c*) Die Option **-a** ist wichtig, damit auch die Einträge mit berücksichtigt werden, die mit einem Punkt beginnen:

```
ls -a | wc -l
```

Aufgabe 2 (10 Punkte)

(*a*) Geben Sie eine kurze Charakterisierung für das, was man in der Unix-Welt unter einem *Filter* versteht!

(*b*) Sie haben ein Programm (z.B. in Oberon) geschrieben, das als ausführbares Programm *a.out* heißt. Sie starten dieses Programm im Vordergrund - und warten, warten, warten - es enthält offenbar eine Endlos-Schleife! Geben Sie an, wie Sie dieses Programm abbrechen!

(*c*) Sie starten das Programm von Teilaufgabe b) erneut, diesmal im Hintergrund. Da Sie wissen, daß dieses Programm einige Zeit laufen wird, arbeiten Sie zunächst weiter. Nach einiger Zeit wollen Sie wissen, ob dieses Programm fertig ist oder noch läuft. Geben Sie ein Unix-Kommando an, mit dem Sie dies herausfinden können!

(*d*) Nachdem Sie festgestellt haben, daß Ihr Programm (von Teilaufgabe c)) immer noch läuft, entschließen Sie sich, es abzubrechen. Geben Sie ein Unix-Kommando an, mit dem Sie dies bewerkstelligen können!

Lösung:

(*a*) Ein Filter ist ein Programm, das von der Standard-Eingabe einliest und die Ausgabe auf der Standard-Ausgabe ausgibt. Filter sind daher besonders geeignet, in einer UNIX-Pipeline eingesetzt zu werden.

(*b*) Mit CTRL-c bzw. dem Zeichen, das bei "stty intr" eingestellt ist.

(*c*) Mit *ps* bzw. mit *jobs* in einer Shell mit Job-Control.

(*d*) Mit *kill*, gefolgt von der Prozess-ID oder gefolgt von einem % und der Job-ID.

Aufgabe 3 (12 Punkte)

Gegeben sei folgender regulärer Ausdruck:

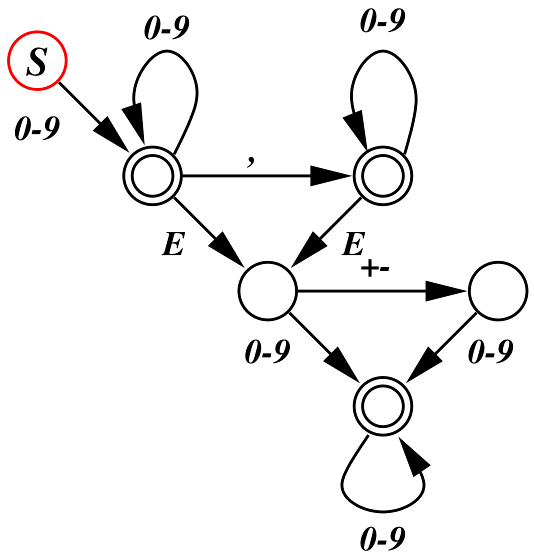
```
[0-9][0-9]*(,[0-9]*)?(E[+-]?[0-9][0-9]*)?
```

(*a*) Bitte geben Sie den zugehörigen endlichen Automaten in graphischer Form an. Pfeile dürfen dabei mehrere Symbole tragen, z.B. 0-9 für die Ziffern 0 bis 9. Markieren Sie bitte den Anfangszustand und die zulässigen Endzustände als solche. Fehlerzustände sind nicht zu berücksichtigen.

(*b*) Bitte kreuzen Sie jeweils an, ob es sich bei den einzelnen Zeichenfolgen um einen Satz der durch den regulären Ausdruck definierten Sprache handelt oder nicht:

Zeichenfolge	Satz? ja	Satz? nein
4711		
23*5+2		
123E4		
623,		
1.0		
755,01E-40		
,234E+10		

(*a*) Alle Zustände mit einem Doppelkreis sind legale Endzustände. Der Anfangszustand ist mit “S” gekennzeichnet.



(*b*)

Zeichenfolge	Satz? ja	Satz? nein
4711	X	
23*5+2		X
123E4	X	
623,	X	
1.0		X
755,01E-40	X	
,234E+10		X

Aufgabe 4 (12 Punkte)

Gegeben sei folgende Datenstruktur eines binären Baumes, der nach Namen sortiert ist (Komponente *name* in *NodeRec*) und für jeden eingetragenen Namen eine lineare Liste von Nachrichten bereithält:

```
TYPE
   TreeOfMessages = POINTER TO TreeOfMessagesRec;
   TreeOfMessagesRec =
      RECORD
         root: Node; (* Wurzel des sortierten binaeren Baumes *)
      END;
   Node = POINTER TO NodeRec;
   NodeRec =
      RECORD
         name: ARRAY 32 OF CHAR; (* nach diesen Namen sortiert *)
         messages: ListOfMessages; (* zugehoerige Nachrichten *)
         left, right: Node; (* siehe Errata! *)
      END;
   ListOfMessages = POINTER TO ListOfMessagesRec;
   ListOfMessagesRec =
      RECORD
         message: ARRAY 128 OF CHAR; (* eine Nachricht *)
         next: ListOfMessages; (* Liste weiterer Nachrichten *)
      END;
```

Schreiben Sie eine Oberon-Prozedur, die für einen gegebenen Namen alle zugehörigen Nachrichten auf dem vorgegebenen Stream ausgibt (jeweils eine Nachricht pro Zeile). Wenn der Name nicht im Baum enthalten ist oder die zugehörige Liste der Nachrichten leer ist, soll nichts ausgegeben werden.

```
PROCEDURE WriteMessages(tree: TreeOfMessages; name: ARRAY OF CHAR;
                        out: Streams.Stream);
```

```
PROCEDURE WriteMessages(tree: TreeOfMessages; name: ARRAY OF CHAR;
                        out: Streams.Stream);

   PROCEDURE WriteMsgs(node: Node; name: ARRAY OF CHAR; out: Streams.Stream);

      PROCEDURE WriteList(messages: ListOfMessages; out: Streams.Stream);
      BEGIN
         WHILE messages # NIL DO
            Write.LineS(out, messages.message);
            messages := messages.next;
         END;
      END WriteList;

   BEGIN (* WriteMsgs *)
      IF node # NIL THEN
         IF node.name = name THEN
            WriteList(node.messages, out);
         ELSIF name < node.name THEN
            WriteMsgs(node.left, name, out);
         ELSE
            WriteMsgs(node.right, name, out);
         END;
      END;
   END WriteMsgs;

BEGIN (* WriteMessages *)
   WriteMsgs(tree.root, name, out);
END WriteMessages;
```

Aufgabe 5 (12 Punkte)

Gegeben sei folgende Grammatik in EBNF mit *Pedigree* als Startsymbol:

```
Pedigree = Name [ Parents ] .
Parents = Person Person .
Person = "(" [ Pedigree ] ")" .
Name = { "A" | .. | "Z" | "a" | .. | "z" } .
```

Stellen Sie bei folgenden Teilaufgaben jeweils fest, ob es sich um einen Satz dieser Grammatik handelt. Wenn ja, sollten Sie eine mögliche Ableitung angeben. Leerzeichen sind dabei zu ignorieren.

(*a*) `Charles`

(*b*) `Charles ()`

(*c*) `Charles (Elizabeth)`

(*d*) `Charles (() Philip)`

(*e*) `Charles () (Philip)`

(*f*) `Charles (Elizabeth) (Philip)`

(*a*) Ist ein Satz. Ableitung:

```
Pedigree
Name
"C" "h" "a" "r" "l" "e" "s"
```

(*b*) Ist kein Satz.

(*c*) Ist kein Satz.

(*d*) Ist kein Satz

(*e*) Ist ein Satz. Ableitung:

```
Pedigree
Name Parents
Name Person Person
Name "(" ")" "(" Pedigree ")"
Name "(" ")" "(" Name ")"
"C" "h" "a" "r" "l" "e" "s" "(" ")" "(" Name ")"
"C" "h" "a" "r" "l" "e" "s" "(" ")" "(" "P" "h" "i" "l" "i" "p" ")"
```

(*f*) Ist ein Satz. Ableitung:

```
Pedigree
Name Parents
Name Person Person
Name "(" Pedigree ")" "(" Pedigree ")"
Name "(" Name ")" "(" Pedigree ")"
Name "(" Name ")" "(" Name ")"
"C" "h" "a" "r" "l" "e" "s" "(" Name ")" "(" Name ")"
"C" "h" "a" "r" "l" "e" "s"
   "(" "E" "l" "i" "z" "a" "b" "e" "t" "h" ")"
   "(" Name ")"
"C" "h" "a" "r" "l" "e" "s"
   "(" "E" "l" "i" "z" "a" "b" "e" "t" "h" ")"
   "(" "P" "h" "i" "l" "i" "p" ")"
```

Aufgabe 6 (12 Punkte)

Gegeben sei folgende Datenstruktur und ein zugehöriger Konstruktor:

```
TYPE
   Pedigree = POINTER TO PedigreeRec;
   PedigreeRec =
      RECORD
         name: ARRAY 32 OF CHAR;
         father, mother: Pedigree;
      END;

PROCEDURE Create(VAR pedigree: Pedigree; name: ARRAY OF CHAR;
                 father, mother: Pedigree);
BEGIN
   NEW(pedigree); COPY(name, pedigree.name);
   pedigree.father := father; pedigree.mother := mother;
END Create;
```

Zeichnen Sie bei den folgenden beiden Teilaufgaben auf, wie die Datenstruktur nach der Ausführung aller Anweisungen der jeweiligen Teilaufgabe aussieht. Die beiden Teilaufgaben sind dabei separat (und nicht etwa aufeinander aufbauend) zu betrachten. Für alle Teilaufgaben gelten folgende Variablenvereinbarungen:

```
VAR franz, marie, hans, melanie: Pedigree;
```

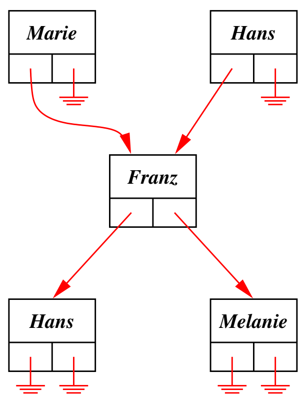
(*a*)

```
Create(franz, "Franz", NIL, NIL);
Create(marie, "Marie", NIL, NIL);
Create(hans, "Hans", franz, marie);
```

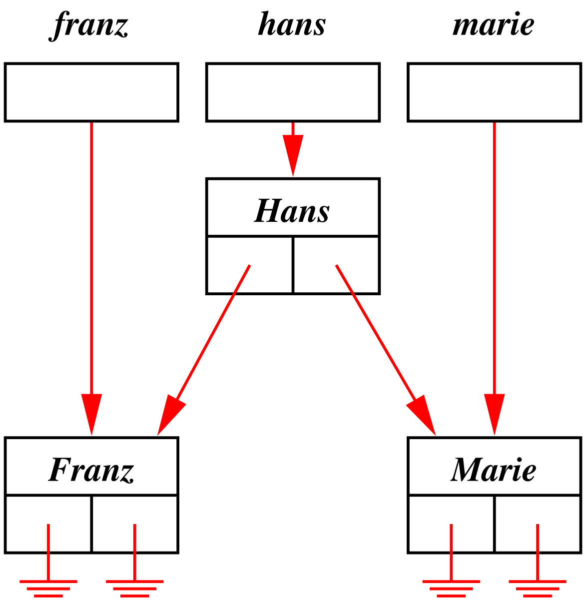
(*b*)

```
Create(franz, "Franz", NIL, NIL);
Create(marie, "Marie", NIL, NIL);
Create(franz, "Franz", franz, marie);
Create(melanie, "Melanie", NIL, NIL);
Create(hans, "Hans", franz, melanie);
```

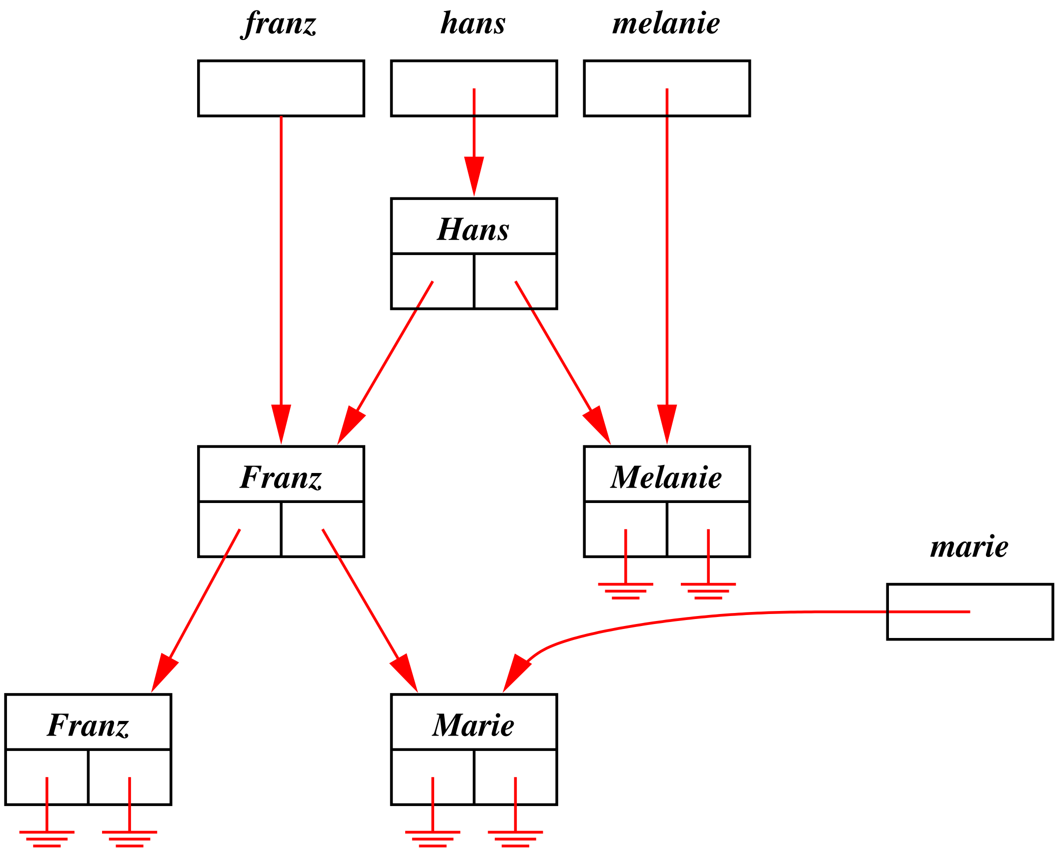
(*c*) Bitte geben Sie die (Oberon-) Anweisungen an, mit der sich folgende konkrete Datenstruktur erzeugen läßt:



(*a*)



(*b*)



(*c*)

```
Create(hans, "Hans", NIL, NIL);
Create(melanie, "Melanie", NIL, NIL);
Create(franz, "Franz", hans, melanie);
Create(marie, "Marie", franz, NIL);
Create(hans, "Hans", franz, NIL);
```

Aufgabe 7 (12 Punkte)

Gegeben sei folgende Datenstruktur und ein zugehöriger Konstruktor:

```
TYPE
   ListElement = POINTER TO ListElementRec;
   ListElementRec =
      RECORD
         next: ListElement;
         age: INTEGER;
      END;

   List = POINTER TO ListRec;
   ListRec =
      RECORD
         head: ListElement;
         average: REAL;
         length: INTEGER;
      END;

PROCEDURE Create(VAR list: List);
BEGIN
   NEW(list); list.head := NIL; list.length := 0;
END Create;
```

Schreiben Sie eine Prozedur `UpdateAverage(list: List)`, die den Mittelwert der Komponenten *age*) der Listenelemente in der Komponente *average* des Listenrecords speichert. Die Berechnung muß *rekursiv* durchgeführt werden.

Lösung:

```
PROCEDURE UpdateAverage(list: List);

   PROCEDURE SumOfElements(element: ListElement) : INTEGER;
   BEGIN
      IF element = NIL THEN
         RETURN 0
      ELSE
         RETURN element.age + SumOfElements(element.next)
      END;
   END SumOfElements;

BEGIN (* UpdateAverage *)
   IF list.length > 0 THEN
      list.average := SumOfElements(list.head) / list.length;
   END;
END UpdateAverage;
```

Aufgabe 8 (12 Punkte)

Gegeben seien folgende Typdefinitionen:

```
TYPE
   Game = POINTER TO GameRec;
   GameRec = RECORD name: ARRAY 80 OF CHAR END;
   CardGame = POINTER TO CardGameRec;
   CardGameRec = RECORD (GameRec) nofCards: INTEGER END;
   Solitaire = POINTER TO SolitaireRec;
   SolitaireRec = RECORD (CardGameRec) version: INTEGER END;
   BlackJack = POINTER TO BlackJackRec;
   BlackJackRec = RECORD (CardGameRec) cardsPlayed: INTEGER; END;
   Chess = POINTER TO ChessRec;
   ChessRec = RECORD (GameRec) location: ARRAY 100 OF CHAR END;
```

(*a*) Zeichnen Sie ein Diagramm, das die Typ-Hierarchie widerspiegelt.

Hinzu kommen folgende Deklarationen:

```
VAR game: Game; cardGame: CardGame; solitaire: Solitaire; chess: Chess;
   blackJack: BlackJack;

PROCEDURE Assign(VAR ptr: Game; game: Game);
BEGIN
   ptr := game;
END Assign;
```

Geben Sie bei folgenden Teilaufgaben jeweils an, ob der jeweilige Programmtext vom Compiler akzeptiert wird und ob er zur Laufzeit fehlerfrei durchläuft. Begründen Sie Ihre Angaben jeweils durch die Angaben der statischen und dynamischen Typen von *ptr* und *game* innerhalb von *Assign*.

(*b*)

```
NEW(solitaire); NEW(blackJack); Assign(solitaire, blackJack);
```

(*c*)

```
NEW(solitaire); NEW(cardGame); Assign(solitaire, cardGame);
```

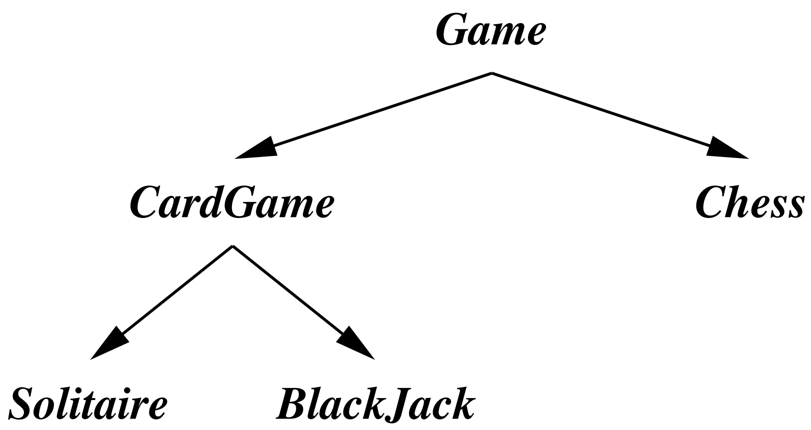
(*d*)

```
NEW(chess); NEW(blackJack); Assign(chess, blackJack);
```

(*e*)

```
NEW(game); NEW(solitaire); Assign(game, solitaire);
```

(*a*)



(*b-e*)

Aufgabe	*ptr*		*game*		Statisch OK?	Zur Laufzeit OK?
	statisch	dynamisch	statisch	dynamisch		
b	*Game*	*Solitaire*	*Game*	*BlackJack*	Ja	Nein
c	*Game*	*Solitaire*	*Game*	*CardGame*	Ja	Nein
d	*Game*	*Chess*	*Game*	*BlackJack*	Ja	Nein
e	*Game*	*Game*	*Game*	*Solitaire*	Ja	Ja

Aufgabe 9 (12 Punkte)

Schreiben Sie eine *rekursive Funktionsprozedur* (in Oberon)
zur Berechnung der n-ten Potenz einer reellen Zahl x mit

$x \in REAL$ und $n \in INTEGER$

Um die Zahl der Multiplikationen gering zu halten,
sind folgende Zusammenhänge zu verwenden:

$$x^n = \frac{1}{x^{-n}} \text{ für } n < 0$$

$$x^n = x^{n/2} \cdot x^{n/2} \text{ für gerade } n$$

$$x^n = x^{(n-1)/2} \cdot x^{(n-1)/2} \cdot x \text{ für ungerade } n$$

Lösung:

```
PROCEDURE Power(x: REAL; n: INTEGER) : REAL;
   VAR
      factor: REAL;
BEGIN
   IF n < 0 THEN
      RETURN 1 / Power(x, -n)
   ELSIF n = 0 THEN
      RETURN 1
   ELSIF ODD(n) THEN
      factor := Power(x, (n-1) DIV 2);
      RETURN x * factor * factor
   ELSE
      factor := Power(x, n DIV 2);
      RETURN factor * factor
   END;
END Power;
```