Programmierung **WS 2000/2001**

Prof. H. Lichter
Th. von der Maßen

Übung 1
Musterlösung

Aufgabe 1.1

a) **Abstraktion**
ein Algorithmus löst i. a. eine **Klasse von Problemstellungen**

Finitheit
- statisch finit: ein Algorithmus besitzt eine **endliche Länge**
- dynamisch finit: während der Abarbeitung darf nur **endlich viel Speicherplatz belegt werden**

Terminierung
- terminierend: nach **endlich vielen Schritten** liegt ein Resultat vor
- sonst **nicht-terminierend**

Determinismus
- deterministisch: zu jedem Zeitpunkt besteht **höchstens eine** Möglichkeit der Fortsetzung
- nicht-deterministisch: an mindestens einer Stelle gibt es eine Wahlmöglichkeit für Fortsetzung

Determiniertheit
- determiniert: bei **gleichen Eingaben** und Startbedingungen wird das gleiche Ergebnis erzielt
- nicht-determiniert: es werden **unterschiedliche Ergebnisse** erzielt

b)

Teilaufgabe	Antwort
i	falsch
ii.	richtig
iii.	richtig
iv.	falsch

Aufgabe 1.2

```
WENN Automat in Betrieb DANN
     SOLANGE NICHT(geforderter Betrag erreicht) UND (noch Geld)
     TUE Geld einwerfen;
     WENN geforderter Betrag erreicht DANN
          WENN Limonade vorhanden DANN
               Knopf für Limonade betätigen;
               WENN Flasche ausgeworfen DANN
                    Flasche entnehmen;
               SONST reklamieren
               WENN geforderter Betrag überschritten DANN
                    Wechselgeld entnehmen;
               SONST Fertig;
          SONST
               Geldrückgabehebel betätigen;
               Geld entnehmen;
     SONST
          Geldrückgabehebel betätigen;
          Geld entnehmen;
SONST anderen Automaten suchen.
```

Aufgabe 1.3

a) $S \xrightarrow{S \to 011} 011$

$S \xrightarrow{S \to A} A \xrightarrow{A \to 0A11} 0A11 \xrightarrow{A \to 0A11} 00A1111 \xrightarrow{A \to \varepsilon} 001111$

$S \xrightarrow{S \to A} A \xrightarrow{A \to 0A11} 0A11 \xrightarrow{A \to 0A11} 00A1111 \xrightarrow{A \to 0A11} 000A111111 \xrightarrow{A \to \varepsilon} 000111111$

b) $L(G) = \{ w \in (0^n 1^{2n}) \mid n \geq 0 \}$
Es gilt hier: $n \geq 0$ da durch die Ableitungsschritte $S \xrightarrow{S \to A} A \xrightarrow{A \to \varepsilon} \varepsilon$ auch das leere Wort erzeugt werden kann.

c) G' = {N', T, P', S}
N' = {S}; T ={ 0, 1 }
P': S → 0S11 | ε

Aufgabe 1.4

a) Kürzestes Wort: aaa

b) G = {N, T, P, {S}}
N = {S, X}; T = {a, b}
P: S → XaXaXaX
X → bX | ε

Musterlösung

Übung 2

Aufgabe 2.1: Syntax, Semantik

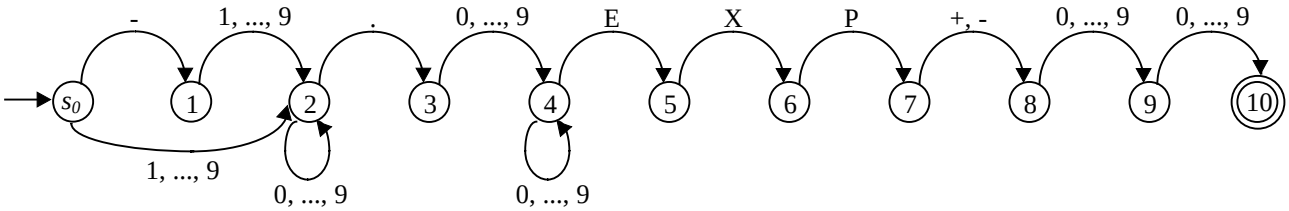
a) *Syntax*: Die Syntax einer Programmiersprache S ist die Definition aller in S zulässigen Aussagen, die in einer Sprache formuliert werden können.
Semantik: Die Semantik einer Sprache S ist die Definition der den zulässigen Aussagen zugeordneten Bedeutungen. Syntaktisch falsche Aussagen haben keine Semantik. Auch syntaktisch korrekte Aussagen haben nicht immer eine Semantik.
Somit ist Syntax ein formaler Begriff, Semantik ein inhaltlicher.
Die Syntax einer Programmiersprache läßt sich ohne Rücksicht auf eine Semantik definieren, d.h. die Syntax läßt sich ohne eine dahinterstehende Semantik definieren. Jedoch macht die Syntax einer Programmiersprache alleine (d.h. ohne Semantik) wenig Sinn. Ein Programm sagt nichts aus ohne eine Semantik. Es ist lediglich ein formales Konstrukt.
Um hingegen die Semantik einer Programmiersprache zu definieren, ist die vorherige Definition einer Syntax notwendig, da durch die Semantik den syntaktischen Konstrukten (bzw. einem Teil dieser) eine Bedeutung zugewiesen wird.

b) Gleiche Syntax impliziert *nicht* gleiche Semantik.
Man betrachte z.B. eine if-Anweisung der Form `if` Bedingung `then` Aktion `endif` . Zunächst kann man dieser die “intuitive” Semantik zuordnen. D. h. wenn die Bedingung (wobei diese auch einer Syntax genügen muß, und die Erfülltheit von der Semantik abhängt) erfüllt ist, dann soll die Aktion (welche ebenfalls die Regeln der Syntax erfüllen muß und die Ausführung von der Semantik abhängt) ausgeführt werden. Man könnte der if-Anweisung jedoch auch folgende Semantik zuordnen: Wenn die Bedingung nicht erfüllt ist, dann soll die Aktion ausgeführt werden.
Die Liste der Möglichkeiten für die Semantik läßt sich beliebig fortsetzen. Die Syntax ist also die gleiche, aber die Semantik nicht.

c) Syntaktisch korrekt bedeutet lediglich, daß ein Programm den Regeln der Syntax genügt. Korrektheit ist jedoch eine inhaltliche Bedingung, die die syntaktische Korrektheit voraussetzt.
Genauer bedeutet *Korrektheit*, daß ein Programm bestimmte Eigenschaften besitzt (festgelegt in den Anforderungen bzw. bei der Vorgabe, welches Problem zu lösen ist). Korrektheit bedeutet also, daß ein Algorithmus auch tatsächlich (für eine beliebige Eingabe) das vorgegebene Problem löst. Durch *syntaktische Korrektheit* ist dies nicht garantiert. Z.B. betrachte man zwei verschiedene Probleme (verschieden bedeutet hier, daß mindestens eine Eingabe existiert, auf die nach A bzw. B verschiedene Antwortengegeben werden müssen) A und B sowie Programme P_A und P_B, welche dieses Problem jeweils korrekt lösen (P_A für A und P_B für B - es wird vorausgesetzt, daß solche existieren). Dann sind P_A und P_B, zwar syntaktisch korrekt aber P_A ist nicht korrekt für B, und P_B ist nicht korrekt für A.

Aufgabe 2.2: Endliche Automaten

Der endliche Automat mit folgendem Zustandsdiagramm leistet das Gewünschte:



Durch den von den Zuständen s_0, 1 und 2 induzierten Teil wird die Zahl vor dem Dezimalpunkt beschrieben. Der Übergang in den Zustand 3 ist nur durch das Lesen des Dezimalpunktes möglich. Eine Transition aus Zustand 3 ist nur durch das Lesen einer Ziffer möglich, denn nach dem Dezimalpunkt muß eine Ziffer folgen.
Um den Endzustand (hier gibt es genau einen Endzustand, dieser ist 10) zu erreichen, müssen die Zustände 5, 6 und 7 in dieser Reihenfolge durchlaufen werden, was nur durch die Folge „EXP“ möglich ist. Ein Übergang aus Zustand 7 ist nur möglich durch Lesen von „+“ oder „-“. Schließlich müssen genau zwei Ziffern folgen, damit das Wort den Bedingungen genügt. Daher kann vom Zustand 8 der Endzustand genau dann erreicht werden, wenn genau zwei Ziffern gelesen werden.

Aufgabe 2.3: EBNF, Syntaxdiagramme

a) Folgende EBNF leistet das Gewünschte:

Pfadname	=	letter “:“ “\“ {dirspec} filename.
rootsymbol	=	“\“.
dirspec	=	basename “\“.
filename	=	basename “.“ extension.
basename	=	namechar {namechar}.
extension	=	namechar {namechar}
namechar	=	(letter | digit | “_“ | “^“ | “$“ | “!“ | “#“ | “%“ | “&“ | “-“ | “{“ | “}“ |“(“ | “)“ | “@“ | “´“ | “`“)
letter	=	(a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v |w| x | y | z | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z)
digit	=	(0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9)

(x_1 | x_2 | ... | x_n) ist die Abkürzung für (x_1 | (x_2 | (... |x_n) ...))

Die Definition von *Pfadname* beschreibt die grobe Struktur eines Pfadnamens wie vorgegeben. Er besteht aus dem Buchstaben für das Dateisystem, gefolgt von einem Doppelpunkt, der Spezifikation des Verzeichnisses und dem Dateinamen. Die Spezifikation des Verzeichnisses ist eine endliche Folge von Verzeichnisnamen, welche jeweils durch „\“ getrennt sind. Am Ende steht ebenfalls „\“. Die übrigen Definitionen definieren die benutzten Namen, welche im wesentlichen endliche Buchstaben - (und Sonderzeichen-) Folgen sind.

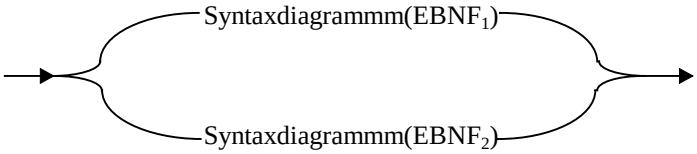
b) Der Algorithmus arbeitet rekursiv über den Aufbau der EBNF. Eine Fallunterscheidung nach dem Konstrukt auf der obersten Ebene wird durchgeführt. Es wird jeweils ein Syntaxdiagramm angegeben, in das die Syntaxdiagramme der EBNFs, die auf den tieferen Ebenen auftreten, wie jeweils angegeben einzusetzen sind. Hierbei bezeichnet Syntaxdiagramm(EBNF) das Syntaxdiagramm zu EBNF, wie es durch den Algorithmus bestimmt wurde.
Der folgende Algorithmus leistet das Gewünschte:

Wenn die EBNF die Form Name = EBNF' besitzt, dann ist Syntaxdiagramm(EBNF):

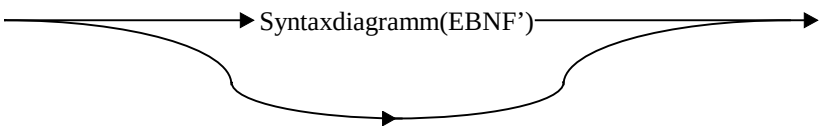
Name

Syntaxdiagramm(EBNF')

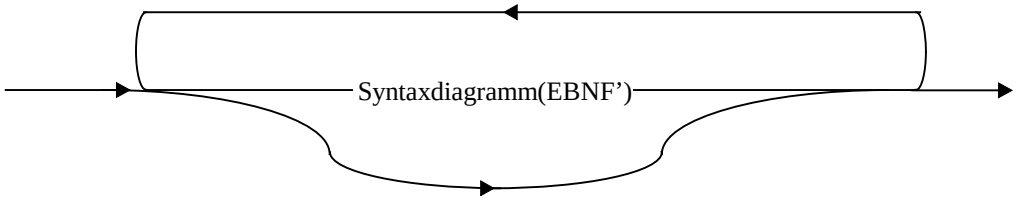
($EBNF_1$| $EBNF_2$) besitzt, dann ist Syntaxdiagramm(EBNF):



[EBNF'] besitzt, dann ist Syntaxdiagramm(EBNF):



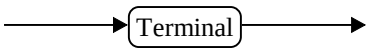
{EBNF'} besitzt, dann ist Syntaxdiagramm(EBNF):



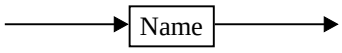
EBNF'.besitzt, dann ist Syntaxdiagramm(EBNF):

Syntaxdiagramm(EBNF')

„Terminal" besitzt, wobei Terminal für ein beliebiges Terminal steht, dann ist Syntaxdiagramm(EBNF):



Name besitzt, wobei Name die Bezeichnung für eine andere EBNF ist, dann ist
Syntaxdiagramm(EBNF):



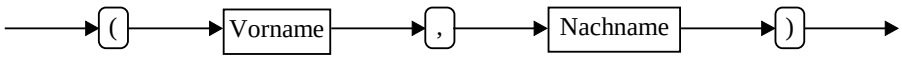
Erläuterung zu dem Algorithmus beispielhaft für den Fall ($EBNF_1$| $EBNF_2$):
Genau eine der Alternativen $EBNF_1$, $EBNF_2$ muß stehen. Genau dieses ist auch im zugehörigen Syntaxdiagramm der Fall. Abgesehen von den möglichen Pfaden innerhalb von Syntaxdiagramm($EBNF_1$) und Syntaxdiagramm($EBNF_2$), gibt es genau zwei mögliche Pfade durch das Syntaxdiagramm: Durch Syntaxdiagramm($EBNF_1$) und durch Syntaxdiagramm($EBNF_2$). Dies entspricht der EBNF.

Rekursion ist hierbei *nicht notwendig*. Der obige Algorithmus kann einfach wiei folgt in einen nicht-rekursiven Algorithmus überführt werden:
Bestimme (zur eingegebenen EBNF) die Menge aller Teil-EBNFs.
Wende die oben angegebenen Regeln zunächst auf die „kleinsten“ EBNFs (dies sind Namen und Terminale) an. Speichere die Ergebnisse ab und gehe hier zur nächst höheren Ebene (die Menge aller Teil-EBNFs, deren Teil-EBNFs nur die EBNFs der vorherigen Ebene und sie selbst sind und welche noch nicht in der Ebene darunter liegen) über. Wende die Regeln auf diese an (da die Teil-EBNFs bereits in Syntaxdiagramme überführt wurden, findet hierbei kein rekursiver Aufruf statt).
Verfahre wie oben beschrieben mit allen Ebenen, bis eine Ebene erreicht ist, zu welcher es keine höhere Ebene mehr gibt (eine solche existiert, da die Menge der Teil-EBNFs endlich ist).

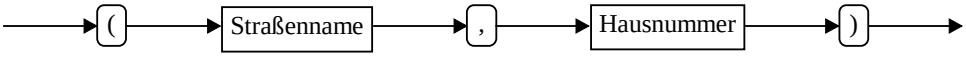
c)

Adresse

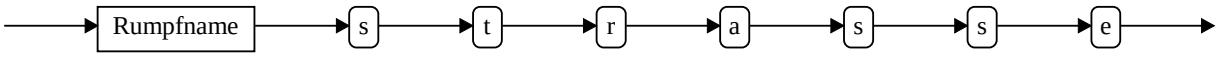
Name



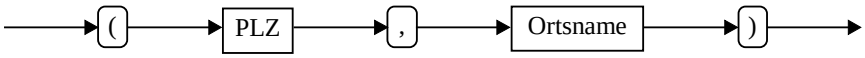
Straße



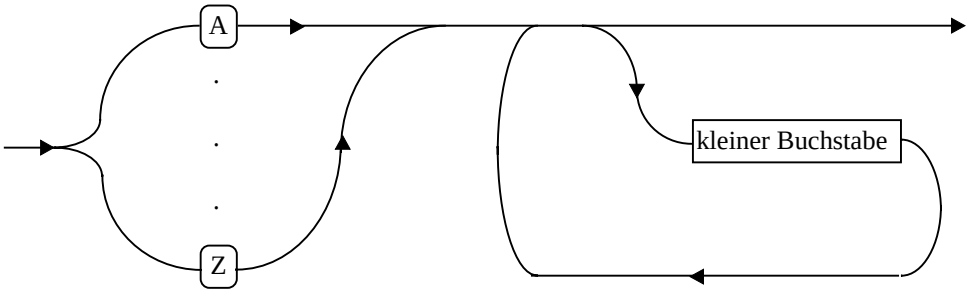
Straßenname



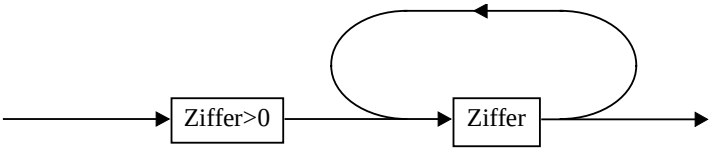
Ort



Vorname, Nachname, Rumpfname, Ortsname

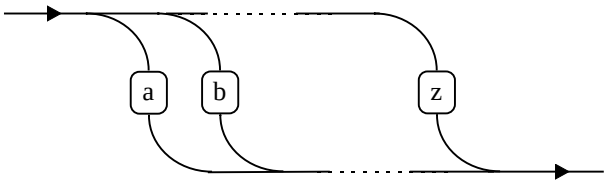


Hausnummer

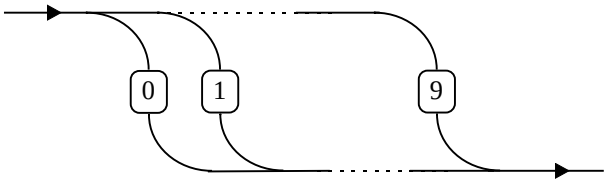


PLZ

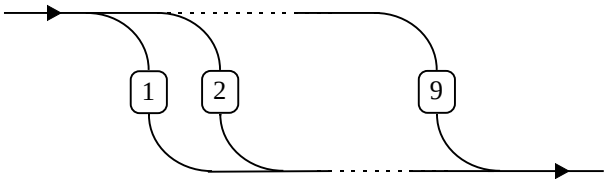
kleiner Buchstabe



Ziffer



Ziffer>0

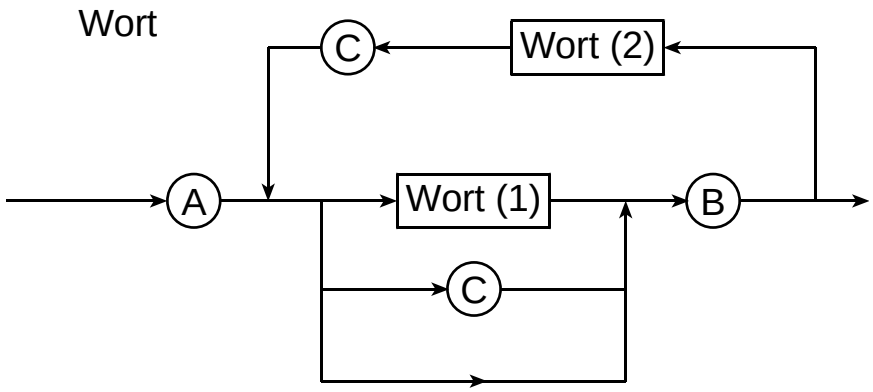


Zunächst wird durch das mit Adresse bezeichnete Syntaxdiagramm die grobe Struktur einer Adresse beschrieben. Die übrigen Syntaxdiagramme definieren die benutzten Komponenten.

Programmierung **WS 2000/2001** **Prof. H. Lichter**
M. Schnizler

Übung 3
Musterlösung

Aufgabe 3.1



Im folgenden ist die jeweils gewählte Rekursion im Syntaxdiagramm (Wort (1) oder Wort (2) als w1 oder w2) abgegrenzt durch senkrechte Striche angedeutet:

a) falsch: A B C ⇒ A B

b) korrekt:
```
  |w1  |
A A C B B
```

c) falsch:
```
                    |w2  |
A B A A C B C B  ⇒  A B A C B C B
```

d) falsch:
```
                                  |w2  |w2 |    |
A B A C C B A B C C B C B  ⇒  A B A C B A B C C B C B
```

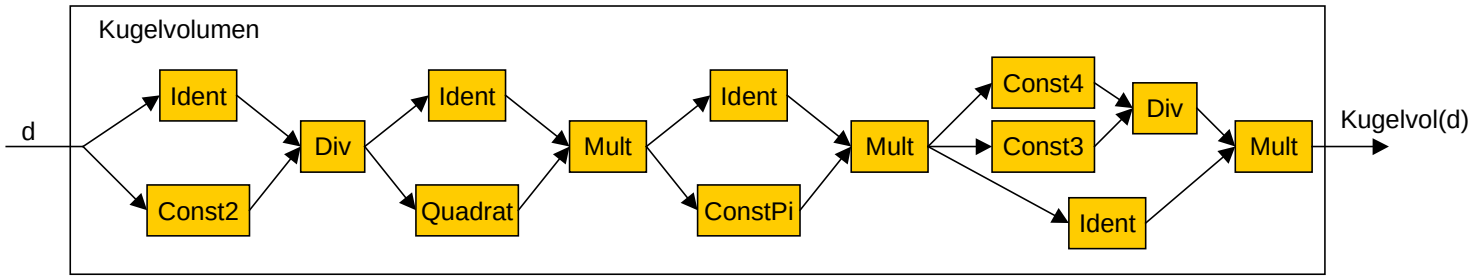
e) korrekt:
```
  |w1  |w2|w1 |  |    |
A A C B A A B B C C B B
```

f) korrekt:
```
    |w2  |w2  |w2|w1  |  |  |    |
A B A C B A B A A C B B C B C C B C B
```

g) korrekt:
```
      |w2|w1|w1  |w2|w1  |  |  |  |  |
A C B A A  A C B A A C B B C B B B C C B
```

Aufgabe 3.2

a) Funktionsnetz:



b) Funktionsausdruck:

```
[Const2, Ident] o Div o [Ident, Quadrat] o Mult o [Ident,
ConstPi] o Mult o[ [Const3, Const4] o Div, Ident] o Mult
```

c) Programm:

```
MODULE Kugel EXPORTS Main;

(* Dieses Programm berechnet rein funktional das Volumen
   einer Kugel.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 14.11.00
   Letzte Aenderung: 14.11.00 *)


IMPORT SIO; (* Importiere notwendige Ein-/Ausgabeoperationen *)


(* Folgende Funktionen ergeben sich aus dem Funktionsnetz. *)

PROCEDURE Ident(a: REAL): REAL =
BEGIN
  RETURN a;
END Ident;

PROCEDURE Const2(a: REAL): REAL =
BEGIN
  RETURN 2.0;
END Const2;

PROCEDURE Const3(a: REAL): REAL =
BEGIN
  RETURN 3.0;
END Const3;

PROCEDURE Const4(a: REAL): REAL =
BEGIN
  RETURN 4.0;
END Const4;
```

```
PROCEDURE ConstPi(a: REAL): REAL =
BEGIN
  RETURN 3.14159265;
END ConstPi;

PROCEDURE Division(a, b: REAL): REAL =
BEGIN
  RETURN a / b;
END Division;

PROCEDURE Mult(a, b: REAL): REAL =
BEGIN
  RETURN a * b;
END Mult;

PROCEDURE Quadrat(a: REAL): REAL =
BEGIN
  RETURN a * a;
END Quadrat;


(* Folgende Funktionen sind notwendig, da Modula-3 keine
   Konstruktion von Funktionsausdruecken unterstuetzt. *)

PROCEDURE Kubik(r: REAL): REAL =
(* Implementiert Ausdruck: [Ident, Quadrat] o Mult *)
BEGIN
  RETURN Mult(Ident(r), Quadrat(r));
END Kubik;

PROCEDURE MalPi(a: REAL): REAL =
(* Implementiert Ausdruck: [Ident, ConstPi] o Mult *)
BEGIN
  RETURN Mult(Ident(a), ConstPi(a));
END MalPi;

PROCEDURE MalVierDrittel(a: REAL): REAL =
(* Implementiert Ausdruck: [[Const4, Const3] o Div, Ident] o Mult *)
BEGIN
  RETURN Mult(Division(Const4(a), Const3(a)), Ident(a));
END MalVierDrittel;


PROCEDURE Kugelvolumen(d: REAL): REAL =
(* Berechnet das Kugelvolumen fuer gegebenen Durchmesser d *)
BEGIN
  RETURN MalVierDrittel(MalPi(Kubik(Division(Ident(d),Const2(d)))));
END Kugelvolumen;

BEGIN
  SIO.PutText("Geben Sie den Kugeldurchmesser (in cm) ein: ");
  SIO.PutReal(Kugelvolumen(SIO.GetReal()));
  SIO.PutLine(" ccm betraegt das Kugelvolumen.");
END Kugel.
```

Probeläufe des Programms für Durchmesser 1 cm, 3 cm und 5 cm:

```
Geben Sie den Kugeldurchmesser (in cm) ein: 1
0.5235988 ccm betraegt das Kugelvolumen.

Geben Sie den Kugeldurchmesser (in cm) ein: 3
14.137168 ccm betraegt das Kugelvolumen.

Geben Sie den Kugeldurchmesser (in cm) ein: 5
65.44985 ccm betraegt das Kugelvolumen.
```

Aufgabe 3.3

```
MODULE Zuege EXPORTS Main;

(* Das Programm berechnet den kuerzesten Zug
   der Gesellschaft Crazy Railways.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 14.11.00
   Letzte Aenderung: 14.11.00 *)

IMPORT SIO; (* Importiere notwendige Ein-/Ausgabeoperationen *)

PROCEDURE Min(x, y : REAL): REAL =
(* Gibt die kleinere der Zahlen x und y zurueck *)
BEGIN
  IF x <= y THEN
    RETURN x;
  ELSE
    RETURN y;
  END;
END Min;

PROCEDURE KuerzesterZug (a : REAL) : REAL =
(* Probiert gemaess der Rekursionsformeln alle Moeglichkeiten,
   solange diese auf einen Wert groesser 0 fuehren. Das Minimum
   der so ermittelten Werte wird zurueckgegeben. *)
BEGIN
  IF a > 33.0 THEN
    RETURN Min (KuerzesterZug((a / 3.0) - 11.0),
                KuerzesterZug(a - 17.0));
  ELSIF a > 17.0 THEN
    RETURN KuerzesterZug(a - 17.0);
  ELSE
    RETURN a;
  END
END KuerzesterZug;

BEGIN
  SIO.PutText("Geben Sie die Laenge (m) des laengsten Zuges ein: ");
  SIO.PutReal(KuerzesterZug(SIO.GetReal()));
  SIO.PutLine(" m ist die Laenge des kuerzesten Zuges.");
END Zuege.
```

Probeläufe des Programms für 15 m, 70 m, 190 m und 1500 m:

```
Geben Sie die Laenge (in m) des laengsten Zuges ein: 15
15 m ist die Laenge des kuerzesten Zuges.

Geben Sie die Laenge (in m) des laengsten Zuges ein: 70
1 m ist die Laenge des kuerzesten Zuges.

Geben Sie die Laenge (in m) des laengsten Zuges ein: 190
0.7777774 m ist die Laenge des kuerzesten Zuges.

Geben Sie die Laenge (in m) des laengsten Zuges ein: 1500
0.12345759 m ist die Laenge des kuerzesten Zuges.
```

Programmierung **WS 2000/2001**

Prof. H. Lichter
Th. von der Maßen

Übung 4
Musterlösung

Aufgabe 4.1

```
MODULE Vokalersetzung EXPORTS Main;

(* Dieses Programm ersetzt in einem Text jeden Vokal durch einen anderen.
   Autor : Th. von der Maßen, RWTH Aachen
   Umgebung : PM-3, Windows 2000
   Erstellt : 06.11.2000
   Letzte Aenderung: 15.11.2000
*)


IMPORT SIO;
IMPORT Text;

(* Diese Hilfsfunktion schneidet das erste Zeichen des übergebenen Textes ab
und liefert
den Rest zurück *)
PROCEDURE Rest(string: TEXT) : TEXT =
BEGIN
  RETURN (Text.Sub(string, 1));
END Rest;


(* Diese Hilfsfunktion gibt das erste Zeichen eines übergebenen Textes als CHAR
zurück *)
PROCEDURE ErstesZeichen(string: TEXT): CHAR =
BEGIN
  RETURN (Text.GetChar(string, 0));
END ErstesZeichen;


(* Diese Funktion überprüft das erste Zeichen des übergebenen Textes. Falls das
Zeichen ein Vokal ist,
wird dieser nach dem geforderten Schema ersetzt. *)
PROCEDURE Ersetze(string: TEXT): TEXT =
BEGIN
  IF Text.Empty(string) THEN
    RETURN ("");
  ELSE
    IF ErstesZeichen(string) = 'a' THEN
      RETURN ("e" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'A' THEN
      RETURN ("E" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'e' THEN
      RETURN ("i" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'E' THEN
      RETURN ("I" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'i' THEN
      RETURN ("o" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'I' THEN
      RETURN ("O" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'o' THEN
      RETURN ("u" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'O' THEN
```

```
      RETURN ("U" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'u' THEN
      RETURN ("a" & Ersetze(Rest(string)));
    ELSIF ErstesZeichen(string) = 'U' THEN
      RETURN ("A" & Ersetze(Rest(string)));
    ELSE
      RETURN (Text.FromChar(ErstesZeichen(string)) & Ersetze(Rest(string)));
    END;
  END;
END Ersetze;


(* Das Hauptmodul liest einen Text von der Tastatur ein und übergibt diesen der
Funktion "Ersetze" *)
BEGIN
  SIO.PutText("Geben Sie einen Text ein: ");
  SIO.PutText("Ergebnis: " & Ersetze(SIO.GetLine()));
  SIO.Nl();
END Vokalersetzung.
```

Ergebnis: Drio Chonisin mot dim Kuntrebess, sotzin eaf dir Stressi and irzeihlin soch wes. Denn kem doi Pulozio and wes ost dinn des? Drio Chonisin mot dim Kuntrebess.

Aufgabe 4.2

a)

f(1)			=1
f(2)			=1
f(3)			=1
f(4)	=	*f(1)-2*f(2)+3*f(3)*	=2
f(5)	=	*f(2)-2*f(3)+3*f(4)*	
	=	*1-2*1+3*(f(1)-2*f(2)+3*f(3))*	=5
f(6)	=	*f(3)-2*f(4)+3*f(5)*	
	=	*1-2*(f(1)-2*f(2)+3*f(3))+3*(f(2)-2*f(3)+3*f(4))*	
	=	*1-2*(1-2*1+3*1)+3*(1-2*1+3*(f(1)-2*f(2)+3*f(3)))*	
	=	*1-2*(1-2*1+3*1)+3*(1-2*1+3*(1-2*1+3*1))*	=12
f(7)	=	*f(4)-2*f(5)+3*f(6)*	
	=	*2-2*(f(2)-2*f(3)+3*f(4))+3*(f(3)-2*f(4)+3*f(5))*	
	=	*2-2*(1-2*1+3*2)+3*(1-2*2+3*(f(2)-2*f(3)+3*f(4)))*	
	=	*2-2*(1-2*1+3*2)+3*(1-2*2+3*(1-2*1+3*2))*	=28
f(8)	=	*f(5)-2*f(6)+3*f(7)*	
	=	*5-2*(f(3)-2*f(4)+3*f(5))+3*(f(4)-2*f(5)+3*f(6))*	
	=	*5-2*(1-2*2+3*5)+3*(2-2*5+3*(f(3)-2*f(4)+3*f(5)))*	
	=	*5-2*(1-2*2+3*5)+3*(2-2*5+3*(1-2*2+3*5))*	=65
f(9)	=	*f(6)-2*f(7)+3*f(8)*	
	=	*12-2*(f(4)-2*f(5)+3*f(6))+3*(f(5)-2*f(6)+3*f(7))*	
	=	*12-2*(2-2*5+3*12)+3*(5-2*12+3*(f(4)-2*f(5)+3*f(6)))*	
	=	*12-2*(2-2*5+3*12)+3*(5-2*12+3*(2-2*5+3*12))*	=151
f(10)	=	*f(7)-2*f(8)+3*f(9)*	
	=	*28-2*(f(5)-2*f(6)+3*f(7))+3*(f(6)-2*f(7)+3*f(8))*	
	=	*28-2*(5-2*12+3*28)+3*(12-2*28+3*(f(5)-2*f(6)+3*f(7)))*	
	=	*28-2*(5-2*12+3*28)+3*(12-2*28+3*(5-2*12+3*28))*	=351

b)

```
MODULE Funktion EXPORTS Main;

(* Dieses Programm berechnet den Funktionswert f(n) für einen gegebenes n
   Autor    : Th. von der Maßen, RWTH Aachen
   Umgebung : PM-3, Windows 2000
   Erstellt : 06.11.2000
   Letzte Aenderung: 15.11.2000
*)

IMPORT SIO;


(* Diese Funktion berechnet die Funktionswert der angegebenen Funktion für die
übergebene Zahl n und gibt diesen zurück *)
PROCEDURE Berechne(n : CARDINAL) : CARDINAL =
BEGIN

  IF (n = 1) OR (n = 2) OR (n = 3) THEN     (* Abbruchbedingung *)
    RETURN 1;
  ELSE
    RETURN (Berechne(n - 3) - 2*Berechne(n - 2) + 3*Berechne(n- 1));
  END;
END Berechne;


(* Das Hauptprogramm liest eine Zahl von der Tastatur ein und übergibt diese an
die Funktion "Berechne" *)
BEGIN
  SIO.PutText("Zahl: ");
  SIO.PutInt(Berechne(SIO.GetInt()));
  SIO.Nl();
END Funktion.
```

n	*f(n)*	*n*	*f(n)*	*n*	*f(n)*
1	1	**10**	351	**19**	696081
2	1	**11**	816	**20**	1618192
3	1	**12**	1897	**21**	3761840
4	2	**13**	4410	**22**	8745217
5	5	**14**	10252	**23**	20330163
6	12	**15**	23833	**24**	47261895
7	28	**16**	55405	**25**	109870576
8	65	**17**	128801		
9	151	**18**	299426		

Aufgabe 4.3

a), b), c) d)

```
MODULE Kubikwurzel EXPORTS Main;

(* Dieses Programm berechnet approximativ die Kubikwurzel für eine gegebene
Zahl unter Anwendung des Newton-Verfahrens
   Autor    : Th. von der Maßen, RWTH Aachen
   Umgebung : PM-3, Windows 2000
   Erstellt : 06.11.2000
   Letzte Aenderung: 15.11.2000
```

```
*)

IMPORT SIO;


(* Diese Funktion berechnet des absoluten Wert der Differenz aus y^3 und x und
gibt diesen zurück *)
PROCEDURE Abstand(y, x : REAL) : REAL =
BEGIN
  RETURN (ABS((y * y * y) - x));
END Abstand;


(* Diese Funktion prüft, ob der Wert des Parameters Abstand kleiner als die
Fehlerschranke e ist. In diesem Fall wird TRUE zurückgegeben, ansonsten FALSE
*)
PROCEDURE GutGenug(abstand, e : REAL) : BOOLEAN =
BEGIN
  IF (abstand < e) THEN
    RETURN TRUE;
    ELSE
      RETURN FALSE;
  END;
END GutGenug;


(* Diese Funktion berechnet einen Näherungswert y als Kubikwurzel aus einer
Zahl x und testet, ob der Näherungswert unter einer vordefinierten Schranke
fehler liegt. *)
PROCEDURE BerechneKubikwurzel(x, y, fehler: REAL) : REAL =
BEGIN
  IF GutGenug(Abstand(y, x), fehler) THEN
    RETURN (y);
  ELSE
    RETURN (BerechneKubikwurzel(x, ((x / (y * y)) + (2.0 * y)) / 3.0, fehler));
  END;
END BerechneKubikwurzel;


(* Diese Prozedur gibt die Kubikwurzel einer Zahl unter Verwendung der Funktion
BerechneKubikwurzel zurück. *)
PROCEDURE Newton(zahl: REAL): REAL =
BEGIN
  SIO.PutText("Die Kubikwurzel lautet: ");
  RETURN (BerechneKubikwurzel(zahl, zahl, 0.0001));
END Newton;


(* Das Hauptprogramm list eine Zahl von der Tastatur ein und übergibt diese an
die Prozedur Newton zur Berechnung der Kubikwurzel aus dieser Zahl. *)
BEGIN
  SIO.PutText("Zahl: ");
  SIO.PutReal(Newton(SIO.GetReal()));
  SIO.Nl();
END Kubikwurzel.
```

x	$\sqrt[3]{x}$	x	$\sqrt[3]{x}$
27	3	**632,5**	8,583943
181	5,656653	**64**	4,0000005
78,961	4,2901344		

Aufgabe 4.4

```
MODULE Lebewesen EXPORTS Main;

(* Dieses Programm berechnet die Anzahl der ausserirdischen Lebewesen in einem
gegeben Jahr n
   Autor    : Th. von der Maßen, RWTH Aachen
   Umgebung : PM-3, Windows 2000
   Erstellt : 06.11.2000
   Letzte Aenderung: 16.11.2000
*)

IMPORT SIO;

(* Diese Hilfsfunktion berechnet die Anzahl der Lebewesen eines Jahres *)
PROCEDURE BerechneAnzahl(anzahl: INTEGER): INTEGER =
BEGIN
  RETURN (anzahl + (anzahl DIV 3));
END BerechneAnzahl;


(* Diese Funktion liefert unter Verwendung der Funktion "BerechneAnzahl", die
Anzahl der Lebewesen eines Jahres. Die Anzahl ergibt sich aus der Anzahl der
Lebewesen des vorherigen Jahres, vermindert um diejenigen, die ihre maximale
Lebensdauer überschritten haben. *)
PROCEDURE AnzahlLebewesen(jahr: INTEGER): INTEGER =
BEGIN
  IF jahr < 0 THEN
    RETURN 0;
  ELSIF jahr = 0 THEN
    RETURN 3;
  ELSE
    RETURN BerechneAnzahl(AnzahlLebewesen(jahr-1) - Neugeborene(jahr-6));
  END;
END AnzahlLebewesen;


(* Diese Hilfsfunktion ermittelt die Neugeborenen Lebewesen eines Jahres *)
PROCEDURE Neugeborene(jahr: INTEGER): INTEGER =
BEGIN
  IF jahr < 0 THEN
    RETURN 0;
  ELSIF jahr = 0 THEN
    RETURN 3;
  ELSE
    RETURN ((AnzahlLebewesen(jahr-1) - Neugeborene(jahr-6)) DIV 3);
  END;
END Neugeborene;

BEGIN
  SIO.PutText("Geben Sie die das Jahr ein: ");
  SIO.PutInt(AnzahlLebewesen(SIO.GetInt()));
  SIO.Nl();
END Lebewesen.
```

Im folgenden seien die Anzahl der Lebewesen nach n Jahren angegeben, wenn im Jahre 0 drei Lebewesen vorhanden waren:

Jahr	Anzahl Lebewesen	Jahr	Anzahl Lebewesen
1	4	**6**	9
2	5	**7**	10
3	6	**8**	12
4	8	**9**	14
5	10	**10**	16

Musterlösung

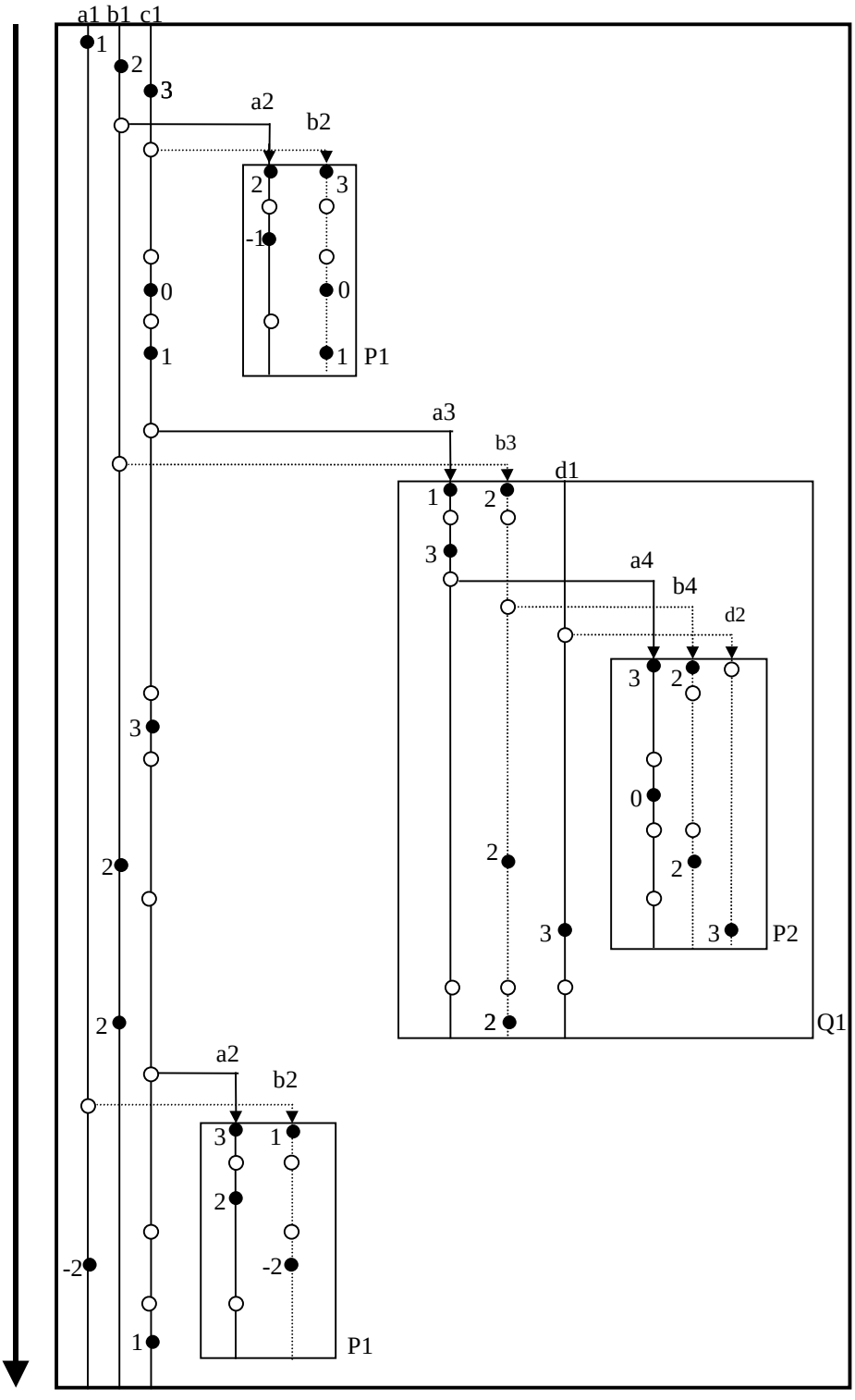
Übung 5

Aufgabe 5.1: Gültigkeitsbereich und Lebensdauer

(Korrigierter) Programmtext mit entsprechend ihrem Gültigkeitsbereich indizierten Variablen und Prozeduren:

```
MODULE M EXPORTS Main;

(* Beginn Gültigkeitsbereich 1 *)

VAR a1, b1, c1: INTEGER;

(* Beginn Gültigkeitsbereich 2 *)

PROCEDURE P1( a2: INTEGER; VAR b2: INTEGER) =
BEGIN
  a2 := a2 - b2;
  b2 := b2 - c1;
  c1 := c1 - a2;
END P1;

(* Ende Gültigkeitsbereich 2 *)

(* Beginn Gültigkeitsbereich 3 *)

PROCEDURE Q1 ( a3: INTEGER; VAR b3: INTEGER)=

(* Beginn Gültigkeitsbereich 4 *)

  PROCEDURE P2( a4: INTEGER; VAR b4: INTEGER; VAR d2: INTEGER) =
  BEGIN
    c1 := c1 + b4;
    a4 := a4 - c1;
    b4 := b4 + a4;
    d2 := c1 + a4;
  END P2;

(* Ende Gültigkeitsbereich 4*)

VAR d1 : INTEGER;
BEGIN
  a3 := a3 + b3;
  P2(a3, b3, d1);
  b3 := b3 + a3 - d1;
END Q1;

(* Ende Gültigkeitsbereich 3 *)

BEGIN
  a1 := 1;  b1 := 2;  c1 := 3;  P1( b1, c1);  Q1( c1, b1);  P1( c1, a1);
END M.
```

Lebensdauerdiagramm:



Aufgabe 5.2: Rekursion, Iteration

a) rekursiv

```
MODULE Potenz EXPORTS Main;

(* Ausgabe von b^n fuer Eingabe b und n
   Autor    : Antje Nowack
   Umgebung : PM3, Windows 95
   Erstellt : 15.11.2000   Letzte Aenderung: 15.11.2000
*)

IMPORT SIO;

(* Die folgenden (Hilfs-)Prozedur wird fuer die Ein- und
Ausgabe benutzt *)

PROCEDURE Eingabe() =

VAR b, n : REAL;

BEGIN
  SIO.PutText("Bitte geben Sie die Basis ein: ");
  b := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Bitte geben Sie den Exponenten ein: ");
  n := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Das Ergebnis lautet: ");
  SIO.PutReal(Potenz(b,n));
  SIO.Nl();
END Eingabe;

(* Die folgende Funktion gibt zu einer natuerlichen Zahl
an, ob sie gerade ist. *)

PROCEDURE IsEven (n : REAL) : BOOLEAN  =
BEGIN
  IF n = 0.0 THEN RETURN TRUE;
  ELSIF n = 1.0 THEN RETURN FALSE;
  ELSE RETURN (IsEven(n-2.0));
  END;
END IsEven ;

(* Die folgende Funktion liefert zu einer
Zahl ihr Quadrat. *)

PROCEDURE Quadrat(x : REAL) : REAL =
BEGIN
  RETURN x*x;
END Quadrat;

(* Die folgende Funktion gibt b^n fuer Eingabe b und n
aus. Sie ist rekursiv definiert. Die Idee dahinter ist
(b^(n/2))^2 = b^n fuer gerade n. Fuer n ungerade gilt:
b^n = b * (b^(n-1)) und n-1 ist gerade. Auf die Potenz
mit (n-1) im Exponenten kann die Regel fuer gerade n
benutzt werden. *)

PROCEDURE Potenz(b, n: REAL) : REAL =
BEGIN
  IF n=0.0 THEN
    RETURN 1.0;
  ELSIF IsEven(n) THEN

(* Rekursiver Aufruf im Fall, dass n gerade ist *)
```

```
    RETURN Quadrat(Potenz(b, n/2.0));
  ELSE

(* Rekursiver Aufruf im Fall, dass n ungerade ist *)

    RETURN (b * Potenz(b, n -1.0));
  END;
END Potenz;

BEGIN
  Eingabe();
END Potenz.
```

Bemerkung: Da $(b^2)^{n/2} = (b^{n/2})^2$ ist es egal, ob Quadrat(Potenz(b, n/2)) oder Potenz(Quadrat(b),n/2) benutzt im rekursiven Aufruf wird.

b) iterativ

```
MODULE Potenz EXPORTS Main;

(* Ausgabe von b^n fuer Eingabe b und n
   Autor    : Antje Nowack
   Umgebung : SRC-Modula-3 rel. 3.6, Windows 95
   Erstellt : 15.11.2000
   Letzte Aenderung: 15.11.2000
*)

IMPORT SIO;

(* Die folgende (Hilfs-)Prozedur wird fuer die Ein- und
Ausgabe benutzt *)

PROCEDURE Eingabe() =

VAR a, b, n : REAL;

BEGIN
  SIO.PutText("Bitte geben Sie die Basis ein: ");
  b := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Bitte geben Sie den Exponenten ein: ");
  n := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Das Ergebnis lautet: ");
  Potenz_iter(b, n, a);
  SIO.PutReal(a);
  SIO.Nl();
END Eingabe;

(*  Die  folgende  iterative  Prozedur  gibt  zu  einer  (natürlichen  Zahl,
einzugeben als Real) Zahl an, ob sie gerade ist. *)

PROCEDURE IsEven (n : REAL) : BOOLEAN  =
BEGIN
  WHILE n >= 0.0 DO
    IF n = 0.0 THEN RETURN TRUE;
    ELSIF n = 1.0 THEN RETURN FALSE;
    ELSE
      n := n – 2.0;
    END;
  END;
END IsEven ;

(* Die folgende Funktion liefert zu einer
Zahl ihr Quadrat. *)
```

```
PROCEDURE Quadrat(x : REAL) : REAL =
BEGIN
  RETURN x*x;
END Quadrat;

(* Die folgende Prozedur gibt b^n fuer Eingabe b und n
aus. Sie arbeitet iterativ. Die Idee dahinter ist
(b^(n/2))^2 = b^n fuer gerade n. Fuer n ungerade gilt:
b^n = b * (b^(n-1)) und n-1 ist gerade. Auf die Potenz
mit (n-1) im Exponenten kann die Regel fuer gerade n
benutzt werden.
Das Ergebnis wird in a akkumuliert.*)

PROCEDURE Potenz_iter(b, n: REAL; VAR a: REAL) =
VAR m, pot : REAL;
BEGIN

(* Hilfsvariablen werden initialisiert. *)
(* m haelt den aktuellen Exponenten *)

  m := n;

(* pot haelt die aktuelle Basis *)

  pot := b;

(* Die Ausgabe wird mit 1 initialisiert. Daher findet keine separate
Abfrage statt, ob der Exponent gleich 0 ist. *)

  a := 1.0;

(* While-Schleife wird im Fall, dass der Exponent nicht 0 ist ausgefuehrt
*)

  WHILE m > 0.0 DO

(* Wenn m gerade ist, wird die entsprechende Regel angewandt *)

    IF IsEven(m) THEN
      pot := Quadrat(pot);
      m := m/2.0;
    ELSE

(* ... ebenso im ungeraden Fall *)

      a := a * pot;
      m := m - 1.0;
    END;
  END;
END Potenz_iter;

BEGIN
 Eingabe();
END Potenz.
```

Aufgabe 5.3: Iteration

```
MODULE Fib EXPORTS Main;

(* Autor: Antje Nowack
   Umgebung: PM3, Windows95
   Erstellt: 22. 11. 2000
   Letzte Aenderung: 22. 11. 2000
*)
```

```
IMPORT SIO;

(* Die folgende Hilfsprozedur wird fuer die Ein-/Ausgabe benutzt. *)

PROCEDURE Eingabe() =

VAR a, b, n : REAL;

BEGIN
  SIO.PutText("Bitte geben Sie a ein: ");
  a := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Bitte geben Sie b ein: ");
  b := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Bitte geben Sie n ein: ");
  n := SIO.GetReal();
  SIO.Nl();
  SIO.PutText("Das Ergebnis lautet: ");
  SIO.PutReal(Fib(a, b, n);
  SIO.Nl();
END Eingabe;



PROCEDURE Fib (a, b, n : CARDINAL ) : CARDINAL =

(* a, b sind die zu uebergebenden Parameter, n ... *)

VAR first, second, result, number: CARDINAL;

(* first gibt das erste Argument auf Stufe number , second das zweite,
result haelt das Ergebnis auf der aktuellen Iterationsstufe *)

  BEGIN
    IF n = 0 THEN RETURN a;
    ELSIF n = 1 THEN RETURN b;
    ELSE

(* Initialisierung von first, second, number *)

      first := a;
      second := b;
      number := 2;

(* Der folgende Teil des Programms wird mehrmals durchlaufen *)
(* In der Schleife wird number pro Durchlauf um 1 erhoeht, first, second,
result entsprechend aktualisiert. Wenn number die Zahl n erreicht, wird der
Schleifendurchlauf beendet. *)

      WHILE number <= n DO
        result := b * first + a * second;
        first := second;
        second := result;
        number := number + 1;
      END;
      RETURN result;
    END;
  END Fib;

BEGIN
  Eingabe();
END Fib.
```

Für a = 1, b =1 liefert das Programm folgende Ergebnisse:

n	f(n)
0	1
1	1
2	2
3	3
4	5
5	8
6	13
7	21
8	34
9	55
10	89

Aufgabe 5.4: call-by-value, call-by-reference

a)

Beim Aufruf der Prozedur wird ein dem Typ des Wertparameters entsprechendes lokales Objekt angelegt. Ist der aktuelle Parameter eine Variable, so entsteht dabei eine Kopie des Parameter-Objekts. Veränderungen des formalen Parameters haben nur lokale Auswirkung. Der aktuelle Parameter bleibt unverändert. Bei Wertparametern bekommt die Prozedur nur den Wert zum Zeitpunkt des Aufrufs mitgeteilt.
Ein Referenzparameter hingegen wird beim Aufruf durch einen Verweis auf den aktuellen Parameter ersetzt. Jede Änderung des formalen Parameters ist direkt im aktuellen Parameter wirksam. Die Kommunikation ist in beiden Richtungen möglich.

Die (unüberlegte) Verwendung von Referenzparametern bringt meist Verständnisprobleme mit sich, da Änderungen potenziell Auswirkungen auf nicht offensichtlich zusammenhängende Argumente haben. Werden Referenzparameter in einer Funktion benutzt, so können hier außerhalb der Funktion Zustandsänderungen durchgeführt werden (vg. Seiteneffekt). Dies zerstört dies Lokalität einer Funktion. Die Änderung an Parametern in Prozeduren ist häufig eine Fehlerquelle, die schwer zu finden ist.

b)

```
MODULE Berechnung EXPORTS Main;
(* Autor: Antje Nowack
   Umgebung: PM3, Windows95
   Erstellt: 22. 11. 2000
   Letzte Aenderung: 22. 11. 2000
*)
IMPORT SIO, Text;


PROCEDURE Eingabe() =

VAR vokal, konsonant, gross, klein, doppel: CARDINAL;
    satz : TEXT;

BEGIN
  vokal := 0;
```

```
  konsonant := 0;
  gross := 0;
  klein := 0;
  doppel := 0;
  SIO.PutText("Geben Sie einen Satz ein: ");
  satz := SIO.GetLine();
  Berechnung (satz, vokal, konsonant, gross, klein, doppel);
  SIO.Nl();
END Eingabe;



(* Die folgende Prozedur berechnet zu einem eingegebenen Text die Anzahl
der eingegebenen Vokale, Konsonanten, Grossbuchstaben, Kleinbuchstaben,
Doppellaute. Die berechnenten befinden sich nach ausfuehrung der
Prozedur in vokal, konsonant, gross, klein, doppel *)

PROCEDURE Berechnung (string: TEXT; VAR vokal, konsonant, gross, klein,
doppel: CARDINAL ) =
  BEGIN

(* Gehe alle Zeichen des eingegebenen Textes durch. *)

    FOR i := 0 TO Text.Length(string)-1 DO

(* Fallunterscheidung nach Auswirkung der Zeichen auf das Ergebnis. *)

     CASE Text.GetChar(string, i) OF

(* Grossbuchstaben und Vokale und kein Doppellaut moeglich *)

       | 'I', 'O', 'U' =>
           vokal := vokal + 1;
           gross := gross +1;

(* Grossbuchstabe und Vokal und Doppellaute "Ai", "Au"  moeglich *)

       | 'A' =>
           vokal := vokal + 1;
           gross := gross +1;

(* Wenn auf das 'A' ein 'i' folgt und somit ein Doppellaut vorliegt ...*)

           IF i < Text.Length(string)-1 AND Text.GetChar(string, i+1)= 'i'
           THEN
             doppel := doppel + 1;

(* Wenn auf das 'A' ein 'u' folgt und somit ein Doppellaut vorliegt ...*)

           ELSIF i < Text.Length(string)-1 AND Text.GetChar(string, i+1)=
           'u' THEN
             doppel := doppel + 1;
           END;

(* Grossbuchstabe und Vokal und Doppellaut "Ei" moeglich *)

      | 'E' =>
           vokal := vokal + 1;
           gross := gross +1;

(* Wenn auf das 'E' ein 'i' folgt und somit ein Doppellaut vorliegt ...*)

           IF i < Text.Length(string)-1 AND Text.GetChar(string, i+1)= 'i'
           THEN
             doppel := doppel + 1;
           END;
```

```
(* Kleinbuchstaben und Vokale und kein Doppellaut moeglich *)

      | 'i', 'o', 'u' =>
          vokal := vokal + 1;
          klein := klein +1;

(* Kleinbuchstabe und Vokal und Doppellaute "ai", "au"  moeglich *)

      | 'a' =>
          vokal := vokal + 1;
          klein := klein +1;

(* Wenn auf das 'a' ein 'i' folgt und somit ein Doppellaut vorliegt ...*)

          IF i < Text.Length(string)-1 AND Text.GetChar(string, i+1)= 'i'
          THEN
            doppel := doppel + 1;

(* Wenn auf das 'a' ein 'u' folgt und somit ein Doppellaut vorliegt ...*)
          ELSIF i <= Text.Length(string)-2 AND Text.GetChar(string, i+1)=
          'u' THEN
            doppel := doppel + 1;
          END;

(* Grossbuchstabe und Vokal und Doppellaut "ei" moeglich *)

       | 'e' =>
          vokal := vokal + 1;
          klein := klein +1;

(* Wenn auf das 'e' ein 'i' folgt und somit ein Doppellaut vorliegt ...*)

          IF i < Text.Length(string)-1 AND Text.GetChar(string, i+1)= 'i'
          THEN
            doppel := doppel + 1;
          END;

(* Kleinbuchstaben und Konsonanten *)
       | 'b', 'c', 'd', 'f', 'g', 'h', 'j', 'k', 'l', 'm', 'n', 'p', 'q',
         'r', 's', 't', 'v', 'w', 'x', 'y', 'z' =>
          klein := klein + 1;
          konsonant := konsonant +1;

(* Grossbuchstaben und Konsonanten *)

       | 'B', 'C', 'D', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P', 'Q',
         'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z' =>
          gross := gross + 1;
          konsonant := konsonant +1;

(* in allen anderen Faellen hat das Zeichen keinen Einfluss auf das
Ergebnis *)

       ELSE
     END;
    END;
    SIO.Nl();
    SIO.PutText("Anzahl der Kleinbuchstaben im Satz: ");
    SIO.PutInt(klein);
    SIO.Nl();
    SIO.PutText("Anzahl der Grossbuchstaben im Satz: ");
    SIO.PutInt(gross);
    SIO.Nl();
    SIO.PutText("Anzahl der Konsonanten im Satz: ");
    SIO.PutInt(konsonant);
    SIO.Nl();
    SIO.PutText("Anzahl der Vokale im Satz: ");
```

```
    SIO.PutInt(vokal);
    SIO.Nl();
    SIO.PutText("Anzahl der Doppellaute im Satz: ");
    SIO.PutInt(doppel);
  END Berechnung;

BEGIN
  Eingabe();
END Berechnung.
```

Testergebnisse:

Der erste Beispielsatz liefert folgendes Ergebnis:

Anzahl der Kleinbuchstaben im Satz: 43
Anzahl der Grossbuchstaben im Satz: 3
Anzahl der Konsonanten im Satz: 26
Anzahl der Vokale im Satz: 20
Anzahl der Doppellaute im Satz: 2

Der zweite Beispielsatz liefert folgendes Ergebnis:

Anzahl der Kleinbuchstaben im Satz: 37
Anzahl der Grossbuchstaben im Satz: 3
Anzahl der Konsonanten im Satz: 25
Anzahl der Vokale im Satz: 15
Anzahl der Doppellaute im Satz: 2

Programmierung **WS 2000/2001** **Prof. H. Lichter**
M. Schnizler

Übung 6
Musterlösung

Aufgabe 6.1 a), b) und c)

```
MODULE Matrizen EXPORTS Main;

(* Dieses Programm multipliziert zwei (5 x 5) Matrizen und
   gibt die Ergebnismatrix aus.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 23.11.00
   Letzte Aenderung: 23.11.00 *)

IMPORT SIO; (* Importiere notwendige Ein-/Ausgabeoperationen *)
IMPORT Fmt; (* Importiere Operationen, um Ausgabe zu formatieren *)

CONST N = 5; (* Konstante für den maximalen Index n *)

(* Notwendige Datentypen für eine (n x n) Matrix *)
TYPE Zeile  = [1 .. N];
     Spalte = [1 .. N];
     Matrix = ARRAY Zeile, Spalte OF REAL;

(* Deklaration der Matrizen/Initialisierung mit Feldaggregat *)
VAR a := Matrix {ARRAY Zeile OF REAL {1.0, 2.0, 3.0, 4.0, 5.0},
                 ARRAY Zeile OF REAL {1.5, 2.5, 3.5, 4.5, 5.5},
                 ARRAY Zeile OF REAL {1.3, 2.3, 3.3, 4.3, 5.3},
                 ARRAY Zeile OF REAL {1.7, 2.7, 3.7, 4.7, 5.7},
                 ARRAY Zeile OF REAL {1.9, 2.9, 3.9, 4.9, 5.9}};
    b := Matrix {ARRAY Zeile OF REAL {2.0, 3.0, 4.0, 5.0, 6.0},
                 ARRAY Zeile OF REAL {2.5, 3.5, 4.5, 5.5, 6.5},
                 ARRAY Zeile OF REAL {2.3, 3.3, 4.3, 5.3, 6.3},
                 ARRAY Zeile OF REAL {2.7, 3.7, 4.7, 5.7, 6.7},
                 ARRAY Zeile OF REAL {2.9, 3.9, 4.9, 5.9, 6.9}};

PROCEDURE MatrixMult(a, b: Matrix): Matrix =
(* Multipliziert die gegebenen Matrizen a und b, liefert die
   Ergebnismatrix als Rückgabewert *)
VAR c: Matrix;
BEGIN
  FOR i := FIRST(Zeile) TO LAST(Zeile) DO
    FOR j := FIRST(Spalte) TO LAST(Spalte) DO
      c [i, j] := 0.0;
      FOR k := FIRST(Spalte) TO LAST(Spalte) DO
        c [i, j] := c [i, j] + a[i, k] * b[k, j];
      END;
    END;
  END;
  RETURN c;
END MatrixMult;
```

```
PROCEDURE MatrixAusgeben(a: Matrix) =
(* Gibt die komplette Matrix elementweise aus *)
BEGIN
  FOR i := FIRST(Zeile) TO LAST(Zeile) DO
    FOR j := FIRST(Spalte) TO LAST(Spalte) DO
      SIO.PutText(MatElementFormat(a [i, j])); SIO.PutText("  ");
    END;
    SIO.Nl();
  END;
END MatrixAusgeben;

PROCEDURE MatElementFormat(r: REAL): TEXT =
(* Formatiere die REAL-Zahl mit Fixpunktnotation und 3 Nachkomma-
   stellen rechtsbuendig in einem festen Feld mit 8 Stellen *)
(* War in der Aufgabe NICHT verlangt! *)
BEGIN
  RETURN Fmt.Pad(Fmt.Real(r, Fmt.Style.Fix, 3),
                 8, ' ', Fmt.Align.Right);
END MatElementFormat;

BEGIN
  (* Ausgeben der zwei zu multiplizierenden Matrizen *)
  MatrixAusgeben(a);
  SIO.PutLine("                          *");
  MatrixAusgeben(b);
  SIO.PutLine("                          =");
  (* Ausgabe des Resultats *)
  MatrixAusgeben(MatrixMult(a, b));
END Matrizen.
```

Ergebnis des Programmlaufs:

```
   1.000     2.000     3.000     4.000     5.000
   1.500     2.500     3.500     4.500     5.500
   1.300     2.300     3.300     4.300     5.300
   1.700     2.700     3.700     4.700     5.700
   1.900     2.900     3.900     4.900     5.900
                        *
   2.000     3.000     4.000     5.000     6.000
   2.500     3.500     4.500     5.500     6.500
   2.300     3.300     4.300     5.300     6.300
   2.700     3.700     4.700     5.700     6.700
   2.900     3.900     4.900     5.900     6.900
                        =
  39.200    54.200    69.200    84.200    99.200
  45.400    62.900    80.400    97.900   115.400
  42.920    59.420    75.920    92.420   108.920
  47.880    66.380    84.880   103.380   121.880
  50.360    69.860    89.360   108.860   128.360
```

Aufgabe 6.2 a), b) und c)

```
MODULE Termine EXPORTS Main;

(* Dieses Programm realisiert eine simple Terminverwaltung, die
   zuvor eingegebene Termine sortiert ausgibt.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 23.11.00
   Letzte Aenderung: 23.11.00 *)

IMPORT SIO; (* Importiere notwendige Ein-/Ausgabeoperationen *)

CONST MAXTERMIN = 10; (* Maximale Anzahl zu verwaltender Termine *)

TYPE (* Unterbereichtstypen und Typ Uhrzeit fuer die Zeitangabe *)
     Stunden        = [0..23];

     Minuten        = [0..59];

     Uhrzeit        = RECORD
                        stunde: Stunden;
                        minute: Minuten;
                      END;

     (* Typ Termin fuer einzelnen Termin: Uhrzeit/Beschreibung *)
     Termin         = RECORD
                        beschreibung: TEXT;
                        zeit        : Uhrzeit;
                      END;

     (* Typ Terminkalender fuer maximal MAXTERMIN Termine *)
     Terminkalender = ARRAY [1..MAXTERMIN] OF Termin;


PROCEDURE TerminEingeben(): Termin =
(* Eingeben eines einzelnen Termins *)
VAR ergTermin: Termin;
BEGIN
  SIO.PutText("Beschreibung des Termins: ");
  ergTermin.beschreibung := SIO.GetLine();
  ergTermin.zeit  := UhrzeitEingeben();
  RETURN ergTermin;
END TerminEingeben;

PROCEDURE TerminAusgeben(termin: Termin) =
(* Ausgeben eines einzelnen Termins *)
BEGIN
  UhrzeitAusgeben(termin.zeit);
  SIO.PutText(" ");
  SIO.PutText(termin.beschreibung);
END TerminAusgeben;
```

```
PROCEDURE UhrzeitEingeben(): Uhrzeit =
(* Eingeben der Uhrzeit mit Stunden und Minuten *)
VAR ergUhrzeit: Uhrzeit;
    restZeile : TEXT;
BEGIN
  SIO.PutText("Uhrzeit (Stunden): ");
  ergUhrzeit.stunde := SIO.GetInt();
  (* Lesen der Restzeile mit RETURN *)
  restZeile := SIO.GetLine();

  SIO.PutText("Uhrzeit (Minuten): ");
  ergUhrzeit.minute := SIO.GetInt();
  (* Lesen der Restzeile mit RETURN *)
  restZeile := SIO.GetLine();

  RETURN ergUhrzeit;
END UhrzeitEingeben;

PROCEDURE StundenMinutenAusgeben(wert: [0..59]) =
(* Hilfsprozedur, um Stunden und Minuten zweistellig auszugeben *)
BEGIN
  IF wert < 10 THEN SIO.PutText("0"); END;
  SIO.PutInt(wert);
END StundenMinutenAusgeben;

PROCEDURE UhrzeitAusgeben(zeit: Uhrzeit) =
(* Gibt die Uhrzeit im Format Stunde:Minute aus *)
BEGIN
  StundenMinutenAusgeben(zeit.stunde);
  SIO.PutText(":");
  StundenMinutenAusgeben(zeit.minute);
  SIO.PutText(" Uhr");
END UhrzeitAusgeben;

PROCEDURE TerminkalenderAusgeben(termine: Terminkalender;
                           maxIndex: INTEGER; aufsteigend: BOOLEAN) =
(* Gibt den bis zum Eintrag maxIndex gefuellten Terminkalender
   termine abhaengig vom angegebenen BOOLEAN-Parameter aufsteigend
   oder absteigend aus. *)
VAR start, ende, schritt: INTEGER;

BEGIN
  IF aufsteigend THEN
    (* Initialisiere Schleifenwerte fuer aufsteigende Ausgabe *)
    start := FIRST(termine); ende := maxIndex; schritt := 1;
  ELSE
    (* Initialisiere Schleifenwerte fuer absteigende Ausgabe *)
    start := maxIndex; ende := FIRST(termine); schritt := -1;
  END;

  (* Ausgeben der Termine *)
  FOR i := start TO ende BY schritt DO
    TerminAusgeben(termine[i]); SIO.Nl();
  END;
END TerminkalenderAusgeben;
```

```
PROCEDURE TauscheTermin(VAR termin1, termin2: Termin) =
(* Hilfsprozedur, um einen einzelnen Termin zu tauschen *)
VAR speicherTermin: Termin;

BEGIN
  speicherTermin := termin1;
  termin1 := termin2;
  termin2 := speicherTermin;
END TauscheTermin;

PROCEDURE TerminkalenderSortieren(VAR termine: Terminkalender;
                                  maxIndex: INTEGER) =
(* Sortiert den bis maxIndex gefuellten Terminkalender termine in
   aufsteigende Reihenfolge *)
BEGIN
  (* Suche fuer jede Position im Terminkalender, ob sich auf den
     noch folgenden Positionen ein frueherer Termin findet *)
  FOR i := FIRST(termine) TO maxIndex DO
    FOR j := i + 1 TO maxIndex DO
      IF (termine[i].zeit.stunde > termine[j].zeit.stunde)  OR
         ( (termine[i].zeit.stunde = termine[j].zeit.stunde) AND
           (termine[i].zeit.minute > termine[j].zeit.minute) )
      THEN
        (* Tausche den Termin mit dem frueheren Termin *)
        TauscheTermin(termine[i], termine[j]);
      END;
    END;
  END;
END TerminkalenderSortieren;


VAR tKalender  : Terminkalender;
    terminIndex: [0..MAXTERMIN] := 0;

    (* Notwendig fuer die Benutzerinteraktion *)
    auswahl    : CHAR;
    restzeile  : TEXT;

BEGIN
  (* Benutzerinteraktion mittels REPEAT-Schleife *)
  REPEAT

    (* Auswahlmenue ausgeben *)
    SIO.Nl();
    SIO.PutLine("***  Terminkalender  ***");
    SIO.PutLine("(e) Neuen Termin eingeben");
    SIO.PutLine("(+) Aufsteigend sortiert ausgeben");
    SIO.PutLine("(-) Absteigend sortiert ausgeben");
    SIO.PutLine("(q) Beendet das Programm");
    SIO.Nl();

    (* Eingabe der ausgewaehlten Operation *)
    SIO.PutText("Auswahl: ");
    auswahl   := SIO.GetChar();
    restzeile := SIO.GetLine();
    SIO.Nl();
```

```
    (* Ausgewaehlte Operation ausfuehren *)
    CASE auswahl OF
       'e', 'E' => IF terminIndex < MAXTERMIN THEN
                     (* Solange noch Platz im Terminkalender *)
                     INC(terminIndex);
                     tKalender[terminIndex] :=TerminEingeben();

                     (* Terminkalender sofort sortieren *)
                     TerminkalenderSortieren(tKalender,
                                             terminIndex);
                   ELSE
                     SIO.PutLine("Terminkalender leider voll!");
                   END;

    |  '+'      => (* Terminkalender aufsteigend ausgeben *)
                   TerminkalenderAusgeben(tKalender,
                                          terminIndex, TRUE);
    |  '-'      => (* Terminkalender absteigend ausgeben *)
                   TerminkalenderAusgeben(tKalender,
                                          terminIndex, FALSE);
    |  'q', 'Q' => SIO.PutLine("Programmende!");
    ELSE
      SIO.PutLine("Ungueltige Eingabe!");
    END;
  UNTIL (auswahl = 'q') OR (auswahl = 'Q');
END Termine.
```

Probelauf des Programms für drei Termine:

```
***  Terminkalender  ***
(e) Neuen Termin eingeben
(+) Aufsteigend sortiert ausgeben
(-) Absteigend sortiert ausgeben
(q) Beendet das Programm

Auswahl: e

Beschreibung des Termins: Frühstücken
Uhrzeit (Stunden): 6
Uhrzeit (Minuten): 00

***  Terminkalender  ***
...

Auswahl: e

Beschreibung des Termins: Abendessen
Uhrzeit (Stunden): 20
Uhrzeit (Minuten): 30

***  Terminkalender  ***
...

Auswahl: e

Beschreibung des Termins: Mittagessen
Uhrzeit (Stunden): 13
Uhrzeit (Minuten): 00
```

```
***  Terminkalender  ***
...

Auswahl: +

06:00 Uhr Frühstücken
13:00 Uhr Mittagessen
20:30 Uhr Abendessen

***  Terminkalender  ***
...

Auswahl: -

20:30 Uhr Abendessen
13:00 Uhr Mittagessen
06:00 Uhr Frühstücken

***  Terminkalender  ***
...

Auswahl: q

Programmende!
```

Aufgabe 6.3 a), b), c) und d)

```
MODULE Meldeamt EXPORTS Main;

(* Dieses Programm realisiert eine einfache Verwaltung fuer
   Einwohnermeldedaten.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows 2000
   Erstellt        : 23.11.00
   Letzte Aenderung: 23.11.00 *)

IMPORT SIO;  (* Importiere notwendige Ein-/Ausgabeoperationen *)
IMPORT Text; (* Importiere notwendige Operationen für Typ TEXT *)

CONST MAXEINWOHNER = 10; (* Maximal zu verwaltende Einwohner *)

TYPE (* Typ Person umfasst alle Namen einer Person *)
     Person         = RECORD
                        familienname : TEXT;
                        rufname      : TEXT;
                        vorname      : ARRAY [2..5] OF TEXT;
                      END;

     (* Notwendige Unterbereichstypen und Typ fuer Tagesdatum *)
     Kalendertag    = [1..31];
     Kalendermonat  = [1..12];
     Kalenderjahr   = [1800..3000];   (* Ausreichender Zeitraum *)
```

```
    Tagesdatum     = RECORD
                       tag  : Kalendertag;
                       monat: Kalendermonat;
                       jahr : Kalenderjahr;
                     END;

    (* Typ Geburtsdaten besteht aus Geburtsort und -datum *)
    Geburtsdaten   = RECORD
                       ort  : TEXT;
                       datum: Tagesdatum;
                     END;

    (* Typ Geschlecht als Aufzaehlungstyp *)
    Geschlecht     = {maennlich, weiblich};

    (* Notwendige Unterbereichstypen und Typ fuer Adresse *)
    Hausnummer     = [1..1000];
    Postleitzahl   = [0..99999];

    Adresse        = RECORD
                       strasse: TEXT;
                       nummer : Hausnummer;
                       zusatz : TEXT;  (* Appartment, Block, ... *)
                       plz    : Postleitzahl;
                       ort    : TEXT;
                     END;

    (* Typ Familienstand als Aufzaehlungstyp *)
    Familienstand  = {ledig, verheiratet, geschieden, verwitwet};

    (* Typ Einwohner beschreibt einen Gemeldeten komplett *)
    Einwohner      = RECORD
                       person       : Person;
                       geboren      : Geburtsdaten;
                       geschlecht   : Geschlecht;
                       wohnung      : Adresse;
                       familienstand: Familienstand;
                       ehepartner   : Person;
                       anmeldung    : Tagesdatum;
                     END;

    (* Typ Melderegister umfasst MAXEINWOHNER  viele Einwohner *)
    Melderegister  = ARRAY [1..MAXEINWOHNER] OF Einwohner;

(* Prozeduren für die Ein- und Ausgabe der Datenstrukturelemente *)

PROCEDURE PersonEingeben(): Person =
(* Eingeben aller Namen einer Person *)
VAR ergPerson: Person;
    i        : INTEGER;
BEGIN
  SIO.PutText("Familienname: ");
  ergPerson.familienname := SIO.GetLine();
  SIO.PutText("Rufname: ");
  ergPerson.rufname := SIO.GetLine();
```

```
  (* Eingabe von bis zu vier weiteren Vornamen.
     Abbruch, falls vier Namen oder nichts eingegeben wurde. *)
  i := FIRST(ergPerson.vorname) - 1; (* Minus 1, zuerst Inkrement *)

  REPEAT
    INC(i); (* Eingabe des naechsten Vornamens *)

    (* Drucke den Text "i. Vorname: " *)
    SIO.PutText(Text.FromChar(VAL(ORD('0')+i,CHAR) )&". Vorname: ");
    ergPerson.vorname[i] :=SIO.GetLine();
  UNTIL (i = LAST(ergPerson.vorname)) OR
        Text.Equal(ergPerson.vorname[i], "");

  RETURN ergPerson;
END PersonEingeben;

PROCEDURE PersonAusgeben(p: Person) =
(* Ausgeben aller Namen einer Person *)
VAR i: INTEGER;
BEGIN
  SIO.PutText(p.familienname);
  SIO.PutText(", ");
  SIO.PutText(p.rufname);
  (* Weitere Vornamen ausgeben, solange maximale Anzahl
     nicht erreicht und kein leerer Name angetroffen wurde. *)
  i := FIRST(p.vorname);
  WHILE (i <= LAST(p.vorname)) AND
        NOT Text.Equal(p.vorname[i], "") DO
    SIO.PutText(" ");
    SIO.PutText(p.vorname[i]);
    INC(i);
  END;
END PersonAusgeben;

PROCEDURE DatumEingeben(): Tagesdatum =
(* Eingeben eines vollstaendigen Datums *)
VAR ergDatum: Tagesdatum;
BEGIN
  SIO.PutText("Tag: ");
  ergDatum.tag   := UBEingeben(FIRST(Kalendertag),
                               LAST(Kalendertag));
  SIO.PutText("Monat: ");
  ergDatum.monat := UBEingeben(FIRST(Kalendermonat),
                               LAST(Kalendermonat));
  SIO.PutText("Jahr: ");
  ergDatum.jahr  := UBEingeben(FIRST(Kalenderjahr),
                               LAST(Kalenderjahr));
  RETURN ergDatum;
END DatumEingeben;

PROCEDURE DatumAusgeben(dat: Tagesdatum) =
(* Ausgeben eines vollstaendigen Datums *)
BEGIN
  SIO.PutInt(dat.tag); SIO.PutText(".");
  SIO.PutInt(dat.monat); SIO.PutText(".");
  SIO.PutInt(dat.jahr);
END DatumAusgeben;
```

```
PROCEDURE GeburtsdatenEingeben(): Geburtsdaten =
(* Eingeben der kompletten Geburtsdaten *)
VAR ergGeburtsdaten: Geburtsdaten;
BEGIN
  SIO.PutText("Geburtsort: ");
  ergGeburtsdaten.ort   := SIO.GetLine();
  SIO.PutLine("Geburtsdatum: ");
  ergGeburtsdaten.datum := DatumEingeben();
  RETURN ergGeburtsdaten;
END GeburtsdatenEingeben;

PROCEDURE GeburtsdatenAusgeben(gd: Geburtsdaten) =
(* Ausgeben der kompletten Geburtsdaten *)
BEGIN
  SIO.PutText("Geboren: ");
  DatumAusgeben(gd.datum);
  SIO.PutText(", ");
  SIO.PutText(gd.ort);
END GeburtsdatenAusgeben;

PROCEDURE GeschlechtEingeben(): Geschlecht =
(* Eingeben des Geschlechts einer Person *)
VAR ergGeschlecht: Geschlecht := FIRST(Geschlecht);
    c            : CHAR;
    restzeile    : TEXT;
    eingabeOK    : BOOLEAN := FALSE;
BEGIN
  (* Fuehre Abfrage solange durch, bis korrekte Eingabe erfolgt *)
  REPEAT
    SIO.PutText("Geschlecht (m, w): ");
    c := SIO.GetChar();
    restzeile := SIO.GetLine(); (* Liest Restzeichen und RETURN *)
    CASE c OF
      'm', 'M' => ergGeschlecht := Geschlecht.maennlich;
                  eingabeOK := TRUE;
    | 'w', 'W' => ergGeschlecht := Geschlecht.weiblich;
                  eingabeOK := TRUE;
    ELSE
      SIO.PutLine("Falsche Eingabe!");
    END;
  UNTIL eingabeOK;
  RETURN ergGeschlecht;
END GeschlechtEingeben;

PROCEDURE GeschlechtAusgeben(g: Geschlecht) =
(* Ausgeben des Geschlechts einer Person *)
BEGIN
  CASE g OF
    Geschlecht.maennlich => SIO.PutText("maennlich");
  | Geschlecht.weiblich  => SIO.PutText("weiblich");
  END;
END GeschlechtAusgeben;

PROCEDURE AdresseEingeben(): Adresse =
(* Eingeben einer vollstaendigen Adresse *)
VAR ergAdresse: Adresse;
BEGIN
  SIO.PutText("Strasse: ");
```

```
  ergAdresse.strasse := SIO.GetLine();
  SIO.PutText("Hausnummer: ");
  ergAdresse.nummer := UBEingeben(FIRST(Hausnummer),
                                  LAST(Hausnummer));
  SIO.PutText("Zusatz: ");
  ergAdresse.zusatz := SIO.GetLine();
  SIO.PutText("Postleitzahl: ");
  ergAdresse.plz := UBEingeben(FIRST(Postleitzahl),
                               LAST(Postleitzahl));
  SIO.PutText("Ort: ");
  ergAdresse.ort := SIO.GetLine();
  RETURN ergAdresse;
END AdresseEingeben;

PROCEDURE AdresseAusgeben(adr: Adresse) =
(* Ausgeben einer vollstaendigen Adresse *)
BEGIN
  (* Strasse mit Hausnummer und Zusatz in einer Zeile *)
  SIO.PutText(adr.strasse); SIO.PutText(" ");
  SIO.PutInt(adr.nummer); SIO.PutText(" ");
  SIO.PutText(adr.zusatz); SIO.Nl();
  (* Postleitzahl mit Ort in neuer Zeile *)
  PostleitzahlAusgeben(adr.plz); SIO.PutText(" ");
  SIO.PutText(adr.ort);
END AdresseAusgeben;

PROCEDURE FamilienstandEingeben(): Familienstand =
(* Eingeben des Familienstands einer Person *)
VAR ergFamilienstand: Familienstand := FIRST(Familienstand);
    c               : CHAR;
    restzeile       : TEXT;
    eingabeOK       : BOOLEAN := FALSE;
BEGIN
  (* Fuehre Abfrage solange durch, bis korrekte Eingabe erfolgt *)
  REPEAT
    SIO.PutText("Familienstand (l, v, g, w): ");
    c := SIO.GetChar();
    restzeile := SIO.GetLine(); (* Liest Restzeichen und RETURN *)

    CASE c OF
      'l', 'L' => ergFamilienstand := Familienstand.ledig;
                  eingabeOK := TRUE;
    | 'v', 'V' => ergFamilienstand := Familienstand.verheiratet;
                  eingabeOK := TRUE;
    | 'g', 'G' => ergFamilienstand := Familienstand.geschieden;
                  eingabeOK := TRUE;
    | 'w', 'W' => ergFamilienstand := Familienstand.verwitwet;
                  eingabeOK := TRUE;
    ELSE
      SIO.PutLine("Falsche Eingabe!");
    END;
  UNTIL eingabeOK;

  RETURN ergFamilienstand;
END FamilienstandEingeben;
```

```
PROCEDURE FamilienstandAusgeben(famstand: Familienstand) =
(* Ausgeben des Familienstands einer Person *)
BEGIN
  CASE famstand OF
    Familienstand.ledig       => SIO.PutText("ledig");
  | Familienstand.verheiratet => SIO.PutText("verheiratet");
  | Familienstand.geschieden  => SIO.PutText("geschieden");
  | Familienstand.verwitwet   => SIO.PutText("verwitwet");
  END;
END FamilienstandAusgeben;

PROCEDURE EinwohnerEingeben(): Einwohner =
(* Komplett Eingeben eines gemeldeten Einwohners *)
VAR ergEinwohner: Einwohner;
BEGIN
  ergEinwohner.person        := PersonEingeben();
  ergEinwohner.geboren       := GeburtsdatenEingeben();
  ergEinwohner.geschlecht    := GeschlechtEingeben();
  ergEinwohner.wohnung       := AdresseEingeben();
  ergEinwohner.familienstand := FamilienstandEingeben();

  IF ergEinwohner.familienstand = Familienstand.verheiratet THEN
    SIO.PutLine("Ehepartner: ");
    ergEinwohner.ehepartner := PersonEingeben();
  END;
  SIO.PutLine("Anmeldung: ");
  ergEinwohner.anmeldung := DatumEingeben();
  RETURN ergEinwohner;
END EinwohnerEingeben;

PROCEDURE EinwohnerAusgeben(einwohner: Einwohner) =
(* Komplett Ausgeben eines gemeldeten Einwohners *)
BEGIN
  PersonAusgeben       (einwohner.person); SIO.Nl();
  GeburtsdatenAusgeben (einwohner.geboren); SIO.Nl();
  GeschlechtAusgeben   (einwohner.geschlecht); SIO.Nl();
  SIO.Nl();
  AdresseAusgeben      (einwohner.wohnung); SIO.Nl();
  SIO.Nl();
  (* Familienstand und, falls verheiratet, Ehepartner ausgeben *)
  FamilienstandAusgeben(einwohner.familienstand);
  IF einwohner.familienstand = Familienstand.verheiratet THEN
    SIO.PutText(" mit ");
    PersonAusgeben(einwohner.ehepartner);
  END;
  SIO.Nl();
  (* Datum der Anmeldung ausgeben *)
  SIO.PutText("Gemeldet seit ");
  DatumAusgeben(einwohner.anmeldung); SIO.Nl();
END EinwohnerAusgeben;

(* Hilfsprozeduren fuer die Ein- und Ausgabe der Daten *)

PROCEDURE UBEingeben(minimum, maximum: INTEGER): INTEGER =
(* Eingeben einer ganzen Zahl innerhalb des zulaessigen
Unterbereichs *)
VAR ergInteger: INTEGER;
    eingabeOK : BOOLEAN := FALSE;
```

```
BEGIN
  (* Fuehre Abfrage solange durch, bis korrekte Eingabe erfolgt *)
  REPEAT
    ergInteger := SIO.GetInt();
    restzeile  := SIO.GetLine(); (* Liest Restzeichen und RETURN *)
    IF (ergInteger >= minimum) AND (ergInteger <= maximum) THEN
      (* Wert im zulaessigen Bereich *)
      eingabeOK := TRUE;
    ELSE
      SIO.PutText("Wert ungueltig! Bitte nochmal eingeben: ");
    END;
  UNTIL eingabeOK;
  RETURN ergInteger;
END UBEingeben;

PROCEDURE PostleitzahlAusgeben(plz: [0..99999]) =
(* Hilfsprozedur, Postleitzahl immer fuenfstellig ausgeben *)
VAR praefix: TEXT;
BEGIN
  (* Zu druckendes Praefix bestimmen *)
  CASE plz OF
    0..9         => praefix := "0000";
  | 10..99       => praefix := "000";
  | 100..999     => praefix := "00";
  | 1000..9999   => praefix := "0";
  | 10000..99999 => praefix := "";
  END;
  (* Erst Praefix, dann eigentliche Zahl ausgeben *)
  SIO.PutText(praefix);
  SIO.PutInt(plz);
END PostleitzahlAusgeben;

PROCEDURE MelderegisterAusgeben(register: Melderegister;
                                maxIndex: INTEGER) =
(* Gibt bis zum Eintrag maxIndex gefuelltes Melderegister aus *)
BEGIN
  (* Ausgeben aller Einwohner *)
  FOR i := FIRST(register) TO maxIndex DO
    SIO.PutInt(i);
    SIO.PutLine(". Einwohner:");
    EinwohnerAusgeben(register[i]);
    SIO.Nl();
  END;
END MelderegisterAusgeben;

VAR register     : Melderegister;
    registerIndex: [0..MAXEINWOHNER] := 0;
    (* Notwendig fuer Benutzerinteraktion *)
    auswahl      : CHAR;
    restzeile    : TEXT;
BEGIN
  (* Benutzerinteraktion mittels REPEAT-Schleife *)
  REPEAT
    (* Auswahlmenue ausgeben *)
    SIO.Nl();
    SIO.PutLine("***  Einwohnermeldeamt  ***");
    SIO.PutLine("(e) Neuen Einwohner anlegen");
    SIO.PutLine("(a) Alle Einwohner ausgeben");
```

```
    SIO.PutLine("(q) Programm beenden");
    SIO.Nl();

    (* Eingabe der ausgewaehlten Operation *)
    SIO.PutText("Auswahl: ");
    auswahl   := SIO.GetChar();
    restzeile := SIO.GetLine();
    SIO.Nl();

    (* Ausgewaehlte Operation ausfuehren *)
    CASE auswahl OF
       'e', 'E' => IF registerIndex < MAXEINWOHNER THEN
                     (* Solange noch Platz im Meldeamt *)
                     INC(registerIndex);
                     register[registerIndex] := EinwohnerEingeben();
                   ELSE
                     SIO.PutLine("Leider kein Platz mehr frei!");
                   END;
    |  'a', 'A' => MelderegisterAusgeben(register, registerIndex);
    |  'q', 'Q' => SIO.PutLine("Programmende!");
    ELSE
      SIO.PutLine("Ungueltige Eingabe!");
    END;
  UNTIL (auswahl = 'q') OR (auswahl = 'Q');
END Meldeamt.
```

Probelauf des Programms für einen Einwohner:

```
***  Einwohnermeldeamt  ***
(e) Neuen Einwohner anlegen
(a) Alle Einwohner ausgeben
(q) Programm beenden

Auswahl: e

Familienname: Müller
Rufname: Willi
2. Vorname: Hans
3. Vorname:
Geburtsort: Bonn
Geburtsdatum:
Tag: 13
Monat: 2
Jahr: 1956
Geschlecht (m, w): maennlich
Strasse: Bahnstr.
Hausnummer: 4
Zusatz: a
Postleitzahl: 43267
Ort: Musterstadt
Familienstand (l, v, g, w): v
Ehepartner:
Familienname: Müller
Rufname: Erika
2. Vorname: Susanne
3. Vorname:
Anmeldung:
Tag: 15
```

```
Monat: 7
Jahr: 1982

***  Einwohnermeldeamt  ***
...

Auswahl: a

1. Einwohner:
Müller, Willi Hans
Geboren: 13.2.1956, Bonn
maennlich

Bahnstr. 4 a
43267 Musterstadt

verheiratet mit Müller, Erika Susanne
Gemeldet seit 15.7.1982


***  Einwohnermeldeamt  ***
...

Auswahl: q

Programmende!
```

Übung 7
Musterlösung

Aufgabe 7.1

a), b), c)

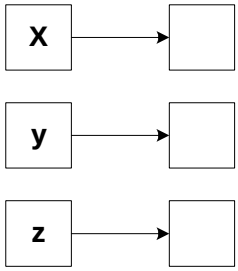
```
MODULE Mengenrelationen EXPORTS Main;

(* Dieses Programm realisiert Mengenrelationen.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3, Windows 2000
   Erstellt        : 29.11.00  Letzte Aenderung: 30.11.00
*)


IMPORT SIO;

(* Deklariere Datentypen *)
TYPE Zeichen = ['a' .. 'z'];
     Buchstaben = SET OF Zeichen;

VAR menge1, menge2 := Buchstaben{};
    alle := Buchstaben{'a' .. 'z'};
    z : CHAR;
    ergebnis : BOOLEAN;


PROCEDURE Ausgabe(m: Buchstaben)=
(* Diese Prozedure gibt alle Elemente einer Menge aus *)
BEGIN
  FOR i:=FIRST(Zeichen) TO LAST(Zeichen) DO
    IF i IN m THEN
      SIO.PutChar(i);
      SIO.PutChar(' ');
    END;
  END;
END Ausgabe;


PROCEDURE Einlesen(VAR m: Buchstaben)=
(* Diese Prozedur liest solange Zeichen von der Tastatur ein und legt diese in
einer Menge ab,
bis '!' eingegeben wird. *)
BEGIN
  REPEAT
    z := SIO.GetChar();
    IF z IN alle THEN
      m := m + Buchstaben{z};
    END;
  UNTIL z = '!';
END Einlesen;


PROCEDURE DruckeMenu()=
(* Diese Prozedur zeigt das Benutzermenü auf dem Bildschirm an. *)
BEGIN
  SIO.PutText("+----------------------+"); SIO.Nl();
```

```
  SIO.PutText("| 1 - Gleichheit       |"); SIO.Nl();
  SIO.PutText("| 2 - Ungleichheit     |"); SIO.Nl();
  SIO.PutText("| 3 - Teilmenge        |"); SIO.Nl();
  SIO.PutText("| 4 - Echte Teilmenge  |"); SIO.Nl();
  SIO.PutText("| 5 - Obermenge        |"); SIO.Nl();
  SIO.PutText("| 6 - Echte Obermenge  |"); SIO.Nl();
  SIO.PutText("| 7 - Beenden          |"); SIO.Nl();
  SIO.PutText("+----------------------+"); SIO.Nl();
  SIO.PutText("Bitte treffen Sie eine Auswahl: ");
END DruckeMenu;

PROCEDURE Gleich(m1, m2: Buchstaben): BOOLEAN =
(* Prüft, ob die Mengen m1 und m2 Mengen gleich sind *)
BEGIN
  RETURN (Teilmenge(m1, m2) AND Teilmenge(m2, m1));
END Gleich;

PROCEDURE Ungleich(m1, m2: Buchstaben): BOOLEAN =
(* Prüft ob die Mengen m1 und m2 ungleich sind *)
BEGIN
  RETURN NOT(Gleich(m1, m2));
END Ungleich;

PROCEDURE Teilmenge(m1, m2: Buchstaben): BOOLEAN =
(* Prüft, ob die Menge m1 eine Teilmenge der Menge m2 ist *)
VAR istGleich: BOOLEAN;
BEGIN
  istGleich := TRUE;
  FOR i:=FIRST(Zeichen) TO LAST(Zeichen) DO
    (* Falls das Element i in m1 ist, dann muss es auch in m2 vorhanden sein *)
    IF i IN m1 THEN
      istGleich := istGleich AND (i IN m2);
    END;
  END;
  RETURN istGleich;
END Teilmenge;

PROCEDURE EchteTeilmenge(m1, m2: Buchstaben): BOOLEAN =
(* Prüft, ob die Menge m1 eine echte Teilmenge der Menge m2 ist *)
BEGIN
  RETURN (Teilmenge(m1, m2) AND Ungleich(m1, m2));
END EchteTeilmenge;

PROCEDURE Obermenge(m1, m2: Buchstaben): BOOLEAN =
(* Prüft, ob die Menge m1 eine Obermenge der Menge m2 ist *)
BEGIN
  RETURN Teilmenge(m2, m1);
END Obermenge;

PROCEDURE EchteObermenge(m1, m2: Buchstaben): BOOLEAN =
(* Prüft, ob die Menge m1 eine echte Obermenge der Menge m2 ist *)
BEGIN
  RETURN EchteTeilmenge(m2, m1);
END EchteObermenge;

BEGIN
  SIO.PutText("Geben Sie bitte die Elemente der 1. Menge ein (Abschliessen mit
'!'):");
  SIO.Nl();
  Einlesen(menge1);
  SIO.PutText("Geben Sie bitte die Elemente der 2. Menge ein (Abschliessen mit
'!'):");
  SIO.Nl();
  Einlesen(menge2);
```

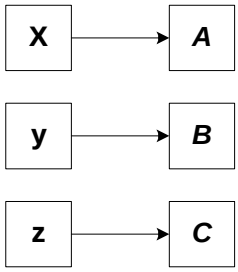
```
  SIO.PutText("Die Elemente der 1. Menge lauten: ");
  Ausgabe(menge1);
  SIO.Nl();
  SIO.PutText("Die Elemente der 2. Menge lauten: ");
  Ausgabe(menge2);
  SIO.Nl();

  ergebnis := TRUE;
  REPEAT
    DruckeMenu();
    (* Umgehung eines Bugs in Windows *)
    z := SIO.GetChar();
    z := SIO.GetChar();
    z := SIO.GetChar();
    CASE z OF
      '1' => ergebnis := Gleich(menge1, menge2); |
      '2' => ergebnis := Ungleich(menge1, menge2); |
      '3' => ergebnis := Teilmenge(menge1, menge2); |
      '4' => ergebnis := EchteTeilmenge(menge1, menge2); |
      '5' => ergebnis := Obermenge(menge1, menge2); |
      '6' => ergebnis := EchteObermenge(menge1, menge2); |
      '7' => SIO.PutText("Ende");
    ELSE
        SIO.PutText("Keine gültige Auswahl !!!");
        SIO.Nl();
    END;
    IF z # '7' THEN
      SIO.PutText("Ergebnis: "); SIO.PutBool(ergebnis); SIO.Nl();
    END;
  UNTIL z = '7';

END Mengenrelationen.
```

Aufgabe 7.2

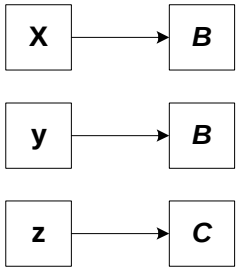
1



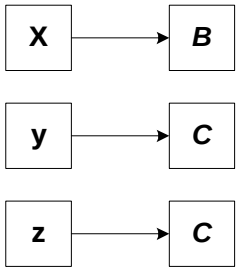
2



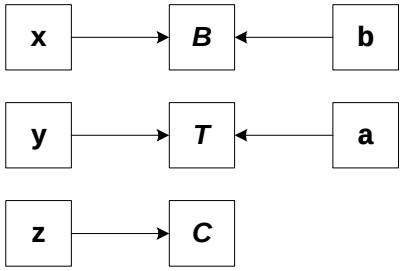
3



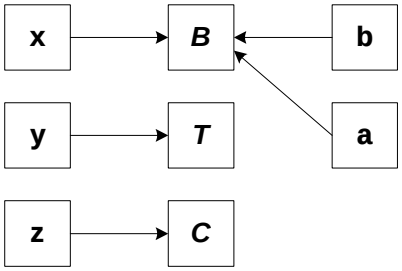
4



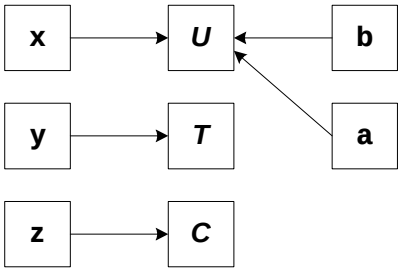
5



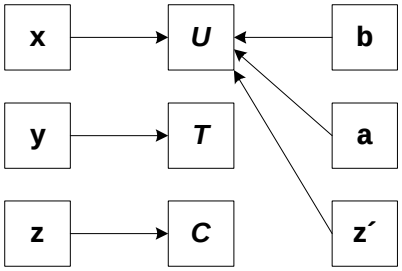
6



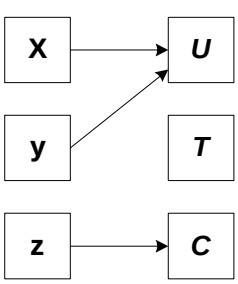
7



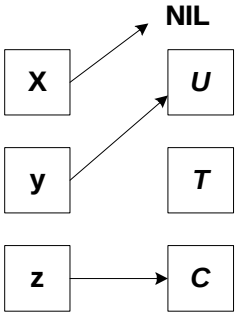
8



9



10



Ausgabe :

```
x = NIL
y = ´U´ ;
z = ´C´ :
```

`z := NEW(CharRef)`	Reservierung von Speicherplatz auf der Halde. Achtung: Der Inhalt z^ ist zwar zugreifbar, aber in einem nicht initalisierten Zustand
`z^ := ´C´`	Der Inhalt z^ ist nun auf das Zeichen ´C´ gesetzt
`x^ := y^`	Der Inhalt y^ wird nach x^ kopiert und ist somit zweimal im Speicher vorhanden, wenn x und y auf verschiedene Speicherbereiche weisen
`a := b`	Die Zeigervariablen verweist nun auf den selben Inhalt wie b.
`x := NIL`	NIL ist ein für jeden Zeigertyp definierter Adresswert und ist so interpretierbar, dass die Varibale x zwar in einem definierten Zustand, ihr aber kein Inhalt zugewiesen ist, also x inhaltsleer ist. Dieser Zustand kann mit `x = NIL` getestet werden.

Aufgabe 7.3

```
MODULE Boxrangliste EXPORTS Main;

(* Dieses Programm dient der Verwaltung einer Boxrangliste.
   Autor           : Moritz Schnizler, Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3, Windows 2000
   Erstellt        : 29.11.00  Letzte Aenderung: 15.12.00
*)


IMPORT SIO;   (* Importiere Operationen fuer die Ein-/Ausgabe *)
IMPORT Text;  (* Importiere Operationen fuer Texte *)


TYPE Name = RECORD
              vorname : TEXT;
              nachname: TEXT;
            END;

     Boxer = RECORD
                name       : Name;
                nation     : TEXT;
                gewicht    : CARDINAL;
              END;

     RanglisteRef = REF RanglistenElement;

     RanglistenElement = RECORD
                           boxer    : Boxer;
                           naechster: RanglisteRef;
                         END;


VAR rangliste    : RanglisteRef; (* Anker der Club-Rangliste *)
    boxer,
    gewinner,
    verlierer    : Boxer;
    auswahl      : CHAR;
    dummy        : TEXT;
```

```
PROCEDURE InitialisiereRL(VAR liste: RanglisteRef)=
(* Initialisiert die Datenstruktur fuer eine Rangliste *)
BEGIN
  liste := NIL; (* Setze Anker der Rangliste auf NIL *)
END InitialisiereRL;


PROCEDURE SucheBoxerRL(liste: RanglisteRef; boxer: Boxer): RanglisteRef=
(* Sucht den gegebenen Boxer in der Rangliste liste und gibt einen Zeiger
   auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der Boxer nicht in der
   Liste vorhanden ist. *)

VAR resRLRef: RanglisteRef;
    gefunden: BOOLEAN;

BEGIN
   gefunden := FALSE;
   resRLRef := liste;
   WHILE (resRLRef # NIL) AND NOT gefunden DO

     (* Pruefe, ob der referenzierte Boxer der gesuchte ist *)
     IF (Text.Equal(resRLRef^.boxer.name.vorname, boxer.name.vorname)) AND
        (Text.Equal(resRLRef^.boxer.name.nachname, boxer.name.nachname)) AND
        (Text.Equal(resRLRef^.boxer.nation,  boxer.nation)) AND
        (resRLRef^.boxer.gewicht = boxer.gewicht)
     THEN
       gefunden := TRUE;
     ELSE
       resRLRef := resRLRef^.naechster;
     END;

   END; (* WHILE *)

   RETURN resRLRef;
END SucheBoxerRL;


PROCEDURE SucheVorgaengerRL(liste: RanglisteRef; boxer: Boxer): RanglisteRef=
(* Sucht den Vorgaenger des gegebenen Boxers in der Rangliste liste und gibt
   einen Zeiger auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der Boxer
keinen
   Vorgaenger hat, d.h. wenn er das erste Element oder nicht in der Liste ist.
*)

VAR vorRLRef, aktRLRef: RanglisteRef;
    gefunden          : BOOLEAN;

BEGIN
   gefunden := FALSE;
   vorRLRef := NIL;    (* Es gibt keinen Vorgaenger des ersten Elements! *)
   aktRLRef := liste;
   WHILE (aktRLRef # NIL) AND NOT gefunden DO

     (* Pruefe, ob der referenzierte Boxer der gesuchte ist *)
     IF (Text.Equal(aktRLRef^.boxer.name.vorname, boxer.name.vorname)) AND
        (Text.Equal(aktRLRef^.boxer.name.nachname, boxer.name.nachname)) AND
        (Text.Equal(aktRLRef^.boxer.nation, boxer.nation)) AND
        (aktRLRef^.boxer.gewicht = boxer.gewicht)
     THEN
       gefunden := TRUE;
     ELSE
       vorRLRef := aktRLRef;
       aktRLRef := aktRLRef^.naechster;
     END;
```

```
   END; (* WHILE *)

   (* Zeiger auf den Vorgaenger ist NIL, wenn Boxer nicht in der Liste ist *)
   IF NOT gefunden THEN vorRLRef := NIL; END;

   RETURN vorRLRef;
END SucheVorgaengerRL;


PROCEDURE DruckeRL(liste: RanglisteRef)=
(* Gibt die als Parameter gegebene Rangliste in absteigender Rangfolge aus *)

VAR ranglisteRef: RanglisteRef;
    rang        : CARDINAL;

BEGIN
  (* Gib Kopfkommentar der Rangliste aus *)
  SIO.PutLine(" *** Box-Rangliste *** ");
  SIO.Nl();

  ranglisteRef := liste;
  rang := 1;
  WHILE (ranglisteRef # NIL) DO

    (* Gib Rang und Daten des Boxers aus *)
    SIO.PutInt(rang); SIO.PutText(". ");
    SIO.PutText(ranglisteRef^.boxer.name.nachname & ", ");
    SIO.PutText(ranglisteRef^.boxer.name.vorname & "   ");
    SIO.PutText("Nation: ");
    SIO.PutText(ranglisteRef^.boxer.nation & "  ");
    SIO.PutText("Gewicht: ");
    SIO.PutInt(ranglisteRef^.boxer.gewicht);
    SIO.Nl();

    (* Inkrementiere Boxer und Rang *)
    ranglisteRef := ranglisteRef^.naechster;
    rang := rang + 1;

  END; (* WHILE *)
END DruckeRL;


PROCEDURE LoescheBoxerRL(VAR liste: RanglisteRef; boxer: Boxer)=
(* Loescht den gegebenen Boxer aus der Rangliste liste, falls vorhanden. *)

VAR ranglisteRef,
    vorgaengerRef: RanglisteRef;

BEGIN
  (* Suche den Boxer in der Liste *)
  ranglisteRef := SucheBoxerRL(liste, boxer);

  IF (ranglisteRef # NIL) THEN

    (* Vorhandenen Boxer aus der Rangliste loeschen *)
    vorgaengerRef := SucheVorgaengerRL(liste, boxer);
    IF vorgaengerRef = NIL THEN
      (* Erstes Element in Rangliste, daher kein Vorgaenger *)
      liste := ranglisteRef^.naechster;
    ELSE
      vorgaengerRef^.naechster := ranglisteRef^.naechster;
    END;
```

```
  ELSE
    SIO.PutLine("FEHLER: Boxer kommt nicht in der Liste vor!");
  END;
END LoescheBoxerRL;


PROCEDURE EinfuegenBoxerRL(VAR liste: RanglisteRef; boxer: Boxer)=
(* Fuegt den gegebenen Boxer am Ende der Rangliste liste ein *)

VAR ranglisteRef, aktRLRef: RanglisteRef;

BEGIN
  (* Pruefe, ob Boxer nicht schon in Rangliste enthalten *)
  ranglisteRef := SucheBoxerRL(liste, boxer);

  IF (ranglisteRef = NIL) THEN

    (* Erzeuge neues Ranglisten-Element fuer den Boxer *)
    ranglisteRef := NEW(RanglisteRef);
    ranglisteRef^.boxer := boxer;
    ranglisteRef^.naechster := NIL;

    (* Fuege Element an Ende der Liste an *)
    IF (liste = NIL) THEN
      (* Liste ist noch leer *)
      liste := ranglisteRef;
    ELSE
      (* Suche Ende der Liste *)
      aktRLRef := liste;
      WHILE (aktRLRef^.naechster # NIL) DO
        aktRLRef := aktRLRef^.naechster;
      END;
      (* Haenge Boxer ans Ende an *)
      aktRLRef^.naechster := ranglisteRef;
    END;
  ELSE
    SIO.PutLine("FEHLER: Boxer ist bereits in der Rangliste!");
  END;
END EinfuegenBoxerRL;


PROCEDURE AKommtHinterBInRL(aRef, bRef: RanglisteRef): BOOLEAN=
(* Prueft, ob das durch Zeiger aRef referenzierte Element hinter dem durch bRef
   referenzierten in der Rangliste vorkommt *)

VAR ranglisteRef: RanglisteRef;

BEGIN
  ranglisteRef := aRef;
  WHILE (ranglisteRef # NIL) AND (ranglisteRef # bRef) DO
    ranglisteRef := ranglisteRef^.naechster;
  END;

  RETURN NOT (ranglisteRef = bRef);
END AKommtHinterBInRL;


PROCEDURE AktualisiereRL(VAR liste: RanglisteRef; gewinner, verlierer: Boxer)=
(* Aendert die Rangfolge in der Rangliste entsprechend dem gegebenen
   Boxergebnis (Gewinner, Verlierer) *)

VAR gewinnerRef, verliererRef      : RanglisteRef;
    vorVerliererRef: RanglisteRef;
```

```
BEGIN
  (* Suche zunaechst Gewinner und Verlierer in der Rangliste *)
  gewinnerRef := SucheBoxerRL(liste, gewinner);
  verliererRef := SucheBoxerRL(liste, verlierer);

  IF (gewinnerRef = NIL) OR (verliererRef = NIL) THEN
    SIO.PutLine("FEHLER: Wenigstens einer der Boxer ist nicht in der
Rangliste!");
  ELSE

    (* Pruefe, ob nicht Gewinner ohnehin vor Verlierer in der Rangliste! *)
    IF NOT AKommtHinterBInRL(gewinnerRef, verliererRef) THEN
      SIO.PutLine("Keine Veraenderung in der Rangliste!");
    ELSE

      (* Fuege Verlierer hinter Gewinner in Rangliste ein *)

      (* Suche die jeweiligen Vorgaenger in der Rangliste *)
      vorVerliererRef := SucheVorgaengerRL(liste, verlierer);

      (* Entferne Verlierer aus ursprünglicher Position *)
      IF ( vorVerliererRef = NIL) THEN
        (* Kein Vorgaenger, da ganz am Anfang der Liste *)
        liste := verliererRef^.naechster;
      ELSE
        vorVerliererRef^.naechster := verliererRef^.naechster;
      END;

      (* Hänge Verlierer hinter Gewinner ein *)
      verliererRef^.naechster := gewinnerRef^.naechster;
      gewinnerRef^.naechster := verliererRef;
    END;
  END;
END AktualisiereRL;


PROCEDURE LeseBoxerEin(nachricht: TEXT): Boxer=
(* Fragt den Benutzer nach den Daten eines Boxers und gibt
   anschliessend einen entsprechenden Boxer zurueck *)

VAR resBoxer     : Boxer;
    dummy          : TEXT;

BEGIN
  SIO.PutLine(nachricht);
  SIO.PutText("Nachname   : ");
  resBoxer.name.nachname := SIO.GetLine();
  SIO.PutText("Vorname    : ");
  resBoxer.name.vorname := SIO.GetLine();
  SIO.PutText("Nation     : ");
  resBoxer.nation := SIO.GetLine();
  SIO.PutText("Gewicht    : ");
  resBoxer.gewicht := SIO.GetInt();

  (* Lies RETURN nach Gewicht aus der Eingabe! *)
  dummy := SIO.GetLine();

  RETURN resBoxer;
END LeseBoxerEin;

PROCEDURE DruckeMenue()=
(* Hilfsprozedur, die das Befehlsmenue ausgibt *)
BEGIN
  SIO.Nl();
```

```
  SIO.PutLine("Z : Zeige Rangliste an");
  SIO.PutLine("F : Fuege einen Boxer ein");
  SIO.PutLine("L : Loesche einen Boxer");
  SIO.PutLine("G : Kampf gemacht");
  SIO.PutLine("E : Exit");
  SIO.Nl();
END DruckeMenue;


BEGIN
  InitialisiereRL(rangliste);
  REPEAT
    DruckeMenue();

    SIO.PutText("Auswahl: ");
    auswahl := SIO.GetChar();
    dummy := SIO.GetLine(); (* Lies RETURN aus dem Eingabepuffer! *)
    SIO.Nl();

    CASE auswahl OF
    |  'Z', 'z' => DruckeRL(rangliste);
    |  'F', 'f' => boxer := LeseBoxerEin("Boxer Einfuegen"); SIO.Nl();
                   EinfuegenBoxerRL(rangliste, boxer);
    |  'L', 'l' => boxer := LeseBoxerEin("Boxer Loeschen"); SIO.Nl();
                   LoescheBoxerRL(rangliste, boxer);
    |  'G', 'g' => gewinner := LeseBoxerEin("Sieger des Spiels"); SIO.Nl();
                   verlierer := LeseBoxerEin("Verlierer des Spiels"); SIO.Nl();
                   AktualisiereRL(rangliste, gewinner, verlierer);
    |  'E', 'e' => (* Nichts machen, gleich ist ohnehin alles zu Ende! *)
    ELSE
      SIO.PutLine("Ungueltiger Befehl!");
    END (* CASE *)

  UNTIL (auswahl = 'e') OR (auswahl = 'E');
END Boxrangliste.
```

Programmierung	WS 2000 / 2001	Prof. H. Lichter A. Nowack

Übung 8

Musterlösung

Aufgabe 8.1:

```
MODULE Zykl_Liste EXPORTS Main;

(* Autor: Antje Nowack
   Umgebung: PM3, Windows95
   Erstellt: 16. 12. 2000
   Letzte Aenderung: 18. 12. 2000
*)

IMPORT SIO;

(*-------------------------Aufgabenteil a-------------------------*)

(* Definition der Datenstruktur. Der Datensatz besteht aus einer
   Zahl vom Typ Integer. Jedes Element besitzt einen Nachfolger und
   einen Vorgaenger. *)

TYPE Link = REF Element;
     Element = RECORD
                 nr : INTEGER;
                 vor, nach : Link;
               END;

(* Die benutzten globalen Variablen werden definiert. kette ist der
   Anker zur aktuellen Liste.
   wahl enthaelt die aktuelle Wahl im Hauptprogramm.
   neu soll beim Einfuegen die einzufuegende Zahl enthalten.
   dummy dient zum Abfangen der Probleme mit SIO.GetInt().
   a,b steuern die Zerstoerung/Beschaedigung der Verkettung einer
   Liste. *)

VAR kette : Link;
    wahl, neu : INTEGER;
    dummy : TEXT;
    a,b : CARDINAL;



PROCEDURE SucheLetzten(kette : Link) : Link =
(* Diese Funktion liefert zu einer Liste den Vorgaenger des Ankers.
*)

(* aktuell enthaelt das aktuell betrachtete Listenelement. *)
VAR aktuell : Link;
  BEGIN
    (* Der Fall der leeren Liste wird nicht betrachtet, da diese
       Funktion nicht mit der leeren Liste aufgerufen wird. *)
    (* Wenn die Liste aus mehr aus einem Element besteht ... *)
```

```
    IF kette.nach # kette THEN
      aktuell := kette^.nach;
      (* Die Schleife wird so lange ausgefuehrt, bis der Vorgaenger
         des Ankers erreicht ist ... *)
      WHILE aktuell^.nach # kette DO
      (* jeweils das naechste Listenelement wird betrachtet ... *)
        aktuell := aktuell^.nach;
      END;
      (* ... und der Vorgaenger des Ankers zurueckgegeben. *)
      RETURN aktuell;
    ELSE
     RETURN kette;
  END;
END SucheLetzten;


PROCEDURE Einfuegen(VAR kette : Link;  neu : INTEGER)=
(* Diese Prozedur fuegt in eine Liste eine Zahl ein. Hierbei wird
   das einzufuegende Element Vorgaenger des Ankers (und Nachfolger
   des urspruenglichen Vorgaengers des Ankers). Der urspruengliche
   Anker bleibt der Anker. *)

VAR elem : Link;
BEGIN
  elem := NEW(Link);
  elem^.nr := neu;
  IF kette = NIL THEN
    elem^.nach := elem;
    elem^.vor := elem;
    kette := elem;
  ELSE
    elem^.nach := kette;
    elem^.vor := SucheLetzten(kette);
    SucheLetzten(kette)^.nach := elem;
    kette^.vor := elem;
  END;
END Einfuegen;


PROCEDURE AusgebenVorwaerts (kette : Link) =
(* Diese Prozedur gibt eine Liste auf dem Bildschirm aus, indem
   zuerst der Inhalt des Ankers ausgegeben und dann jeweils der des
   Nachfolgers des aktuellen Elementes, bis der Vorgaenger des
   Ankers erreicht ist. *)

VAR aktuell : Link;
BEGIN
  IF kette # NIL THEN
    SIO.PutInt(kette^.nr);
    SIO.Nl();
    dummy := SIO.GetLine();
    IF kette.nach # kette THEN
      aktuell := kette^.nach;
      WHILE aktuell # kette DO
        SIO.PutInt(aktuell^.nr);
        SIO.Nl();
        dummy := SIO.GetLine();
        aktuell := aktuell^.nach;
      END;
    END;
```

```
  END;
END AusgebenVorwaerts;


PROCEDURE AusgebenRueckwaerts (kette : Link) =
(* Diese Prozedur gibt eine Liste auf dem Bildschirm aus, indem
   zuerst der Inhalt des Ankers ausgegeben und dann jeweils der des
   Vorgaengers des aktuellen Elementes bis der Nachfolger des Ankers
   erreicht ist. *)

VAR aktuell : Link;
BEGIN
  IF kette # NIL THEN
    SIO.PutInt(kette^.nr);
    SIO.Nl();
    dummy := SIO.GetLine();
    IF kette.vor # kette THEN
      aktuell := kette^.vor;
      WHILE aktuell # kette DO
        SIO.PutInt(aktuell^.nr);
        SIO.Nl();
        dummy := SIO.GetLine();
        aktuell := aktuell^.vor;
      END;
    END;
  END;
END AusgebenRueckwaerts;


PROCEDURE Zerstoeren(VAR kette : Link; a, b : CARDINAL) =
(* Mit Hilfe dieser Prozedur kann die Verkettung einer Liste
   beschaedigt werden. Uebergeben wird die Kette (mit Hilfe eines
   Referenzparameters) und zwei natuerliche Zahlen a und b.
   Der b-te Nachfolger des Ankers wird zum Nachfolgers des a-ten
   Nachfolgers des Ankers. *)

VAR element_a, element_b : Link;
BEGIN
  element_a := kette;
  element_b := kette;
  FOR i := 0 TO a DO
    element_a := element_a^.nach;
  END;
  FOR i := 0 TO b DO
    element_b := element_b^.nach;
  END;
  element_a^.nach := element_b;
END Zerstoeren;

(*-------------------------Aufgabenteil b-------------------------*)


PROCEDURE Test(kette: Link): BOOLEAN=
(* Diese Funktion liefert zu einer doppelt verketteten zyklischen
   Liste einen booleschen Wert. Dieser ist TRUE gdw. die Verkettung
   nicht beschaedigt ist. *)

VAR vorElement, nachElement: Link;
    verkettungOK           : BOOLEAN;
BEGIN
```

```
  (* Zunaechst den Sonderfall der leeren Liste abfangen. *)
  IF kette = NIL THEN
    RETURN TRUE;
  ELSE
    (* Der Test laesst sich auf eine lokale Bedingung reduzieren.
       Wenn NIL auftritt als Nachfolger oder wenn der Vorgaenger des
       Nachfolgers eines Elementes nicht das Element selber ist, so
       ist die Verkettung beschaedigt. Andere Faelle koennen nicht
       auftreten. *)
    nachElement := kette;
    vorElement  := kette^.vor;  (* Nicht leere Kette! *)
    verkettungOK := TRUE;
    (* Ueberpruefung der obigen lokalen Bedingung in einer Schleife
       ... *)
    REPEAT
      IF (vorElement = NIL) OR
         (nachElement # vorElement^.nach) THEN
         verkettungOK := FALSE;
      ELSE
        nachElement := vorElement;
        vorElement := vorElement^.vor;
      END;
    (* ... bis eine Stelle gefunden wurde, die die Beschaedigung der
       Verkettung belegt oder die gesamte Liste durchlaufen wurde
       (ohne eine solche Stelle zu finden). *)
    UNTIL (vorElement = kette^.vor) OR NOT verkettungOK;
    RETURN verkettungOK;
  END;
END Test;

(*-------------------------Hauptprogramm-------------------------*)

(* Steuerung ueber ein Menue *)

BEGIN
  kette := NIL;
  WHILE wahl # 6 DO
    SIO.Nl();
    SIO.PutLine("******************** Menue ********************");
    SIO.Nl();
    SIO.PutLine("Waehlen Sie einen Menuepunkt:");
    SIO.Nl();
    SIO.PutLine("Zahl zur Liste hinzufuegen (1)");
    SIO.Nl();
    SIO.PutLine("Liste vorwaerts ausgeben (2)");
    SIO.Nl();
    SIO.PutLine("Liste rueckwaerts ausgeben (3)");
    SIO.Nl();
    SIO.PutLine("Zerstoeren der Verkettung (4)");
    SIO.Nl();
    SIO.PutLine("Ueberpruefung der Verkettung (5)");
    SIO.Nl();
    SIO.PutLine("Programm beenden (6)");
    SIO.Nl();
    SIO.PutLine("****************** Ende Menue ******************");
    SIO.Nl();
    SIO.PutText("Wahl: ");
    wahl := SIO.GetInt();
    dummy := SIO.GetLine();
```

```
    IF wahl = 1 THEN
      SIO.PutText("Bitte geben Sie die einzufuegende Zahl ein: ");
      neu := SIO.GetInt();
      dummy := SIO.GetLine();
      Einfuegen(kette, neu);

    ELSIF wahl = 2 THEN
      SIO.PutLine("Die Liste sieht vorwaerts wie folgt aus: ");
      AusgebenVorwaerts(kette);
      SIO.PutLine("Hier schliesst sich der Zykel. ");

    ELSIF wahl = 3 THEN
      SIO.PutLine("Die Liste sieht rueckwaerts wie folgt aus: ");
      AusgebenRueckwaerts(kette);
      SIO.PutLine("Hier schliesst sich der Zykel. ");

    ELSIF wahl = 4 THEN
      SIO.PutText("Bitte geben Sie die erste Position an: ");
      a := SIO.GetInt();
      dummy := SIO.GetLine();
      SIO.PutText("Bitte geben Sie die zweite Position an: ");
      b := SIO.GetInt();
      dummy := SIO.GetLine();
      Zerstoeren(kette, a, b);

   ELSIF wahl = 5 THEN
      IF Test(kette) THEN
        SIO.PutLine("Die Verkettung ist NICHT BESCHAEDIGT.");
      ELSE
        SIO.PutLine("Die Verkettung ist BESCHAEDIGT.");
      END;

    ELSIF wahl # 6 THEN
      SIO.PutLine("Keine korrekte Eingabe. Bitte wiederholen.");
    END;
  END;
  SIO.PutLine("Programmende!");
END Zykl_Liste.
```

Probelauf des Programms:

```
************************* Menue *************************

Waehlen Sie einen Menuepunkt:

Zahl zur Liste hinzufuegen (1)

Liste vorwaerts ausgeben (2)

Liste rueckwaerts ausgeben (3)

Zerstoeren der Verkettung (4)

Ueberpruefung der Verkettung (5)

Programm beenden (6)

************************* Ende Menue *************************
```

```
Wahl:
1
Bitte geben Sie die einzufuegende Zahl ein: 1

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1
Bitte geben Sie die einzufuegende Zahl ein: 2

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1
Bitte geben Sie die einzufuegende Zahl ein: 3

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1
Bitte geben Sie die einzufuegende Zahl ein: 4

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1
Bitte geben Sie die einzufuegende Zahl ein: 5

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 2
Die Liste sieht vorwaerts wie folgt aus:
1

2

3

4

5


Hier schliesst sich der Zykel.

************************* Menue ***************
...
************************* Ende Menue ***********

Wahl:3
Die Liste sieht rueckwaerts wie folgt aus:
1

5
```

```
4

3

2

Hier schliesst sich der Zykel.

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 5
Die Verkettung ist NICHT BESCHAEDIGT.

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 4
Bitte geben Sie die erste Position an: 2
Bitte geben Sie die zweite Position an: 4

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 5
Die Verkettung ist BESCHAEDIGT.

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 4
Bitte geben Sie die erste Position an: 2
Bitte geben Sie die zweite Position an: 4

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 5
Die Verkettung ist BESCHAEDIGT.

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 2
Die Liste sieht vorwaerts wie folgt aus:
1

2

3


4
```

```
Hier schliesst sich der Zykel.

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 3
Die Liste sieht rueckwaerts wie folgt aus:
1

5

4

3


2

Hier schliesst sich der Zykel.

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 6
Programmende!
```

Aufgabe 8.2:

```
MODULE Schuelerliste EXPORTS Main;

(* Autor: Antje Nowack
   Umgebung: PM3, Windows95
   Erstellt: 16. 12. 2000
   Letzte Aenderung: 21. 12. 2000
*)

IMPORT SIO, Math;

(*----------------------Aufgabenteil a --------------------------*)

(* Ein Schueler wird durch einen Verbund/Record mit den
   entsprechenden Datentypen realisiert. *)
TYPE Schueler = RECORD
                  Name : TEXT;
                  Vorname : TEXT;
                  Alter : REAL;
                  Note : REAL;
                END;

     (* Mit Hilfe der Datenstruktur SchuelerRef laesst sich eine
        einfach verkettete Liste, in der Schueler abgelegt sind,
        realisieren. *)
     SchuelerRef = REF Element;
     Element = RECORD
                  schueler : Schueler;
```

```
                naechster : SchuelerRef;
            END;

     (* Die folgenden beiden Typen sind Prozedurtypen.
        Operation enthaelt alle zweistelligen Funktionen auf REAL,
        welche einen REAL-Wert zurueckliefern. Dies sind z.B.
        Addition und Multiplikation. *)
     Operation = PROCEDURE (a,b : REAL) : REAL;

     (* Projektion enthaelt alle einstelligen Funktionen, die
        Schueler auf REAL abbilden. Dies sind z.B. Die Funktionen,
        die einen Schueler auf seine Note oder sein Alter abbilden.
     *)
     Projektion = PROCEDURE (schueler : Schueler) : REAL;

     (* Vergleich enthaelt alle zweistelligen Relationen, d.h.
        booleschen Funktionen, auf reellen Zahlen, z.B. die <- und
        die >-Relation. *)
     Vergleich = PROCEDURE (a,b : REAL) : BOOLEAN;

(* ... einige Hilfsvariablen ... *)
VAR schuelerListe : SchuelerRef := NIL;
    schueler : Schueler;
    wahl: INTEGER;
    dummy : TEXT;
    sortiert : SchuelerRef;
    limitAlter, limitNote : REAL;


PROCEDURE Einfuegen(VAR liste : SchuelerRef; neu : Schueler)=
(* Diese Prozedur fuegt einen Schueler in eine Liste ein. Dieser
   wird an den Beginn der Liste gesetzt. Hierbei wird nicht
   ueberpruft, ob der Schueler bereits in der Liste vorhanden ist.
*)

VAR elem : SchuelerRef;
BEGIN
  elem := NEW(SchuelerRef);
  elem^.schueler := neu;
  elem^.naechster := liste;
  liste := elem;
END Einfuegen;


PROCEDURE Ausgeben(liste : SchuelerRef) =
(* Diese Prozedur gibt eine Liste auf dem Bildschirm aus, indem die
   gesamte Liste durchlaufen wird. *)

BEGIN
  IF liste # NIL THEN
    SIO.PutText("Name: ");
    SIO.PutText(liste^.schueler.Name);
    SIO.Nl();
    SIO.PutText("Vorname: ");
    SIO.PutText(liste^.schueler.Vorname);
    SIO.Nl();
    SIO.PutText("Alter: ");
    SIO.PutReal(liste^.schueler.Alter);
    SIO.Nl();
    SIO.PutText("Note: ");
```

```
    SIO.PutReal(liste^.schueler.Note);
    SIO.Nl();
    SIO.Nl();
    SIO.PutText("-------------------------------------------------");
    SIO.Nl();
    dummy := SIO.GetLine();
    SIO.Nl();
    Ausgeben(liste^.naechster);
    SIO.Nl();
  END;
END Ausgeben;


(*----------------------Aufgabenteil b ---------------------------*)


PROCEDURE Alter (schueler : Schueler) : REAL =
(* Diese Funktion bildet einen Schueler auf sein Alter ab. *)

BEGIN
  RETURN schueler.Alter;
END Alter;


PROCEDURE Note (schueler : Schueler) : REAL =
(* Die folgende Funktion bildet einen Schueler auf seine Note ab. *)

BEGIN
  RETURN schueler.Note;
END Note;

PROCEDURE A_groesser_B ( a, b : REAL) : BOOLEAN =
BEGIN
 RETURN a > b;
END A_groesser_B;

PROCEDURE A_kleiner_B ( a, b : REAL) : BOOLEAN =
BEGIN
 RETURN a < b;
END A_kleiner_B;

PROCEDURE StelleSuchen(liste : SchuelerRef; person : Schueler; proj
: Projektion; rel : Vergleich) : SchuelerRef =
(* Mit Hilfe dieser Prozedur wird zu einem Schueler und einer
  Schuelerliste die Stelle gesucht, wo der Schueler eingefuegt
  werden muss. Hierbei wird von einer sortierten Liste gemaess rel
  ausgegangen. *)

VAR stelle, vergleich : SchuelerRef;
BEGIN
  vergleich := liste;
  stelle := liste;
  WHILE (vergleich # NIL) AND NOT rel(proj ( vergleich^.schueler),
proj(person)) DO
    stelle := vergleich;
    vergleich := vergleich^.naechster;
  END;
  RETURN stelle;
END StelleSuchen;
```

```
PROCEDURE Filter( liste : SchuelerRef; proj : Projektion; rel :
Vergleich; limit : REAL; VAR filterListe : SchuelerRef) =
(* Diese Prozedur liefert zu einer einfach verketteten Liste, einer
   Funktion, die einen Schueler auf ein REAL-Zahl abbildet und einer
   REAL-Zahl die Liste, die Liste der Schueler, deren Bild unter der
   Funktion ueber der Zahl liegen. Die Datensaetze der produzierten
   Liste sind Kopien der Original-Liste und diese ist nach Werten
   der uebergebenen Funktion sortiert gemaess rel. *)

VAR kopie : SchuelerRef;
BEGIN
  IF liste # NIL THEN
    (* Speicherplatz reservieren *)
    kopie := NEW(SchuelerRef);
    (* Datensatz des aktuell betrachteten Listenelementes kopieren
     *)
    kopie^.schueler := liste^.schueler;
    (* nur wenn die Grenze ueberschritten wurde, wird der schueler
in die
       resultierende Liste aufgnommen. *)
    IF rel ( proj(kopie^.schueler), limit) THEN
        (* Die erste Operation zum Einfuegen kann nur durchgefuehrt
           werden, wenn die resultierende Liste bereits mindestens
           ein Element enthaelt und der Funktionswert muss kleiner
           sein als der einzufuegende. Dann wird das neue Element an
           der berechneten Stelle eingefuegt. *)
        IF filterListe # NIL AND
          rel (proj(kopie^.schueler),proj(filterListe^.schueler))
        THEN
          kopie^.naechster :=
                       StelleSuchen(filterListe, kopie^.schueler,
                                    proj, rel)^.naechster;
          StelleSuchen(filterListe, kopie^.schueler, proj,
                                         rel)^.naechster := kopie;
        (* Sonst wird das neue Element an den Anfang der bis jetzt
           berechneten resultierenden Liste gesetzt. *)
        ELSE
          kopie^.naechster := filterListe;
          filterListe := kopie;
        END;
    END;
    (* rekursiver Aufruf der Prozedur *)
    Filter(liste^.naechster, proj, rel, limit, filterListe);
  END;
END Filter;


PROCEDURE FilterAlter(liste: SchuelerRef; limit : REAL; VAR ergebnis
: SchuelerRef) =
(* Diese Prozedur wendet die Prozedur Filter an, wobei durch die
   uebergebene Funktion jeder Schueler auf sein Alter projiziert
   wird und > als Vergleichsrelation uebergeben wird. *)

BEGIN
  Filter( liste, Alter, A_groesser_B,limit, ergebnis);
END FilterAlter;
```

```
PROCEDURE FilterNote_schlechter_spaeter(liste: SchuelerRef; limit :
REAL; VAR ergebnis : SchuelerRef) =
(* Diese Prozedur wendet die Prozedur Filter an, wobei durch die
   uebergebene Funktion jeder Schueler auf seine Note projiziert
   wird und < als Vergleichsrelation uebergeben wird. *)

BEGIN
  Filter( liste, Note, A_groesser_B, limit, ergebnis);
END FilterNote_schlechter_spaeter;

PROCEDURE FilterNote_besser_spaeter(liste: SchuelerRef; limit :
REAL; VAR ergebnis : SchuelerRef) =
(* Diese Prozedur wendet die Prozedur Filter an, wobei durch die
   uebergebene Funktion jeder Schueler auf seine Note projiziert
   wird und < als Vergleichsrelation uebergeben wird. *)

BEGIN
  Filter( liste, Note, A_kleiner_B, limit, ergebnis);
END FilterNote_besser_spaeter;


PROCEDURE Filtern_schlechter_spaeter(liste: SchuelerRef; limitAlter,
limitNote : REAL; VAR ergebnis : SchuelerRef) =
(* Diese Prozedur wendet die beiden vorhergehenden Prozeduren an.
   Zuerst diejenige fuer die Filterung nach Alter. Auf die hieraus
   resultierende Liste wird der Filter nach Noten angewandt, welcher
   die Schueler liefert, die schlechter als eine einzugebende Grenze
   enthaelt.*)

VAR zwischenErgebnis : SchuelerRef;
BEGIN
  FilterAlter(liste, limitAlter, zwischenErgebnis);
  FilterNote_schlechter_spaeter(zwischenErgebnis, limitNote,
ergebnis);
END Filtern_schlechter_spaeter;

PROCEDURE Filtern_besser_spaeter(liste: SchuelerRef; limitAlter,
limitNote : REAL; VAR ergebnis : SchuelerRef) =
(* Diese Prozedur wendet die beiden vorhergehenden Prozeduren an.
   Zuerst diejenige fuer die Filterung nach Alter. Auf hieraus
   resultierende Liste wird der Filter nach Noten angewandt, welcher
   die Schueler liefert, die besser als eine einzugebende Grenze
   enthaelt.*)

VAR zwischenErgebnis : SchuelerRef;
BEGIN
  FilterAlter(liste, limitAlter, zwischenErgebnis);
  FilterNote_besser_spaeter(zwischenErgebnis, limitNote, ergebnis);
END Filtern_besser_spaeter;


(*----------------------Aufgabenteil c -------------------------*)


PROCEDURE Anwenden(op : Operation; liste : SchuelerRef; init : REAL)
: REAL =
(* Dieser Prozedur werden eine zweistellige Funktion auf REAL-Werten
   uebergeben, eine Schuelerliste und ein REAL-Wert uebergeben. Die
   Funktion wird sukzessive auf die Noten der Liste angewandt. Der
   resultierende Wert wird zurueckgegeben. Der uebergebene REAL-Wert
```

```
   dient hierbei als Startwert.
   Die sukzessive Anwendung wird durch rekusiven Aufruf der Prozedur
   durchgefuehrt. *)

BEGIN
  IF liste = NIL THEN
    RETURN init;
  ELSE
    init := op(init, liste^.schueler.Note);
    RETURN Anwenden(op, liste^.naechster, init);
  END;
END Anwenden;


PROCEDURE Add(a,b : REAL) : REAL =
(* Addition *)

BEGIN
  RETURN (a+b);
END Add;


PROCEDURE Mult(a,b : REAL) : REAL =
(* Multiplikation *)

BEGIN
  RETURN (a*b);
END Mult;


PROCEDURE Inc(a, b : REAL) : REAL =
(* Inkrementierung des ersten Wertes *)

BEGIN
  RETURN (a + 1.0)
END Inc;


PROCEDURE Arithm_Mittel(liste : SchuelerRef) : REAL =
(* Diese Funktion liefert zu einer Schuelerliste das arithmetische
   Mittel der Noten. *)

BEGIN
  IF liste # NIL THEN
    (* Nach Definition des arithmetischen Mittels wird die Summe der
       Noten durch die Anzahl der Elemente in der Liste dividiert.
       Die Summe wird berechnet durch sukzessives Anwenden der
       Addition. Startwert ist hierbei 0.
       Die Anzahl der Elemente kann hierbei durch sukzessives
       Anwenden der Funktion Inc berechnet werden. Pro Listenelement
       wird um 1 erhoeht. *)
    RETURN (Anwenden(Add, liste, 0.0) / Anwenden(Inc, liste, 0.0));
  ELSE
    (* Bei der leeren Liste wird 0 zurueckgeliefert. *)
    RETURN 0.0;
  END;
END Arithm_Mittel;


PROCEDURE Geom_Mittel(liste : SchuelerRef) : REAL =
```

```
(* Diese Funktion liefert zu einer Schuelerliste das geometrische
   Mittel der Noten. *)

BEGIN
  IF liste # NIL THEN
  (* Nach Definition des geometrischen Mittels wird aus dem Produkt
     der Noten die n-te Wurzel gezogen. n ist hierbei die Anzahl der
     Elemente in der Liste.
     Das Produkt wird berechnet durch sukzessives Anwenden der
     Multiplikation. Startwert ist hierbei 1.
     Die Anzahl der Elemente kann hierbei durch sukzessives Anwenden
     der Funktion Inc berechnet werden. Pro Listenelement wird um 1
     erhoeht.
     Die Verwendung von FLOAT ist notwendig, da die
     Potenzierungfunktion pow aus Math fuer LONGREAL-Werte definiert
     ist und Noten REAL-Werte sind. *)
     RETURN FLOAT( Math.pow ( FLOAT ( Anwenden(Mult, liste, 1.0),
                                      LONGREAL),
                              FLOAT(1.0 / Anwenden(Inc, liste, 0.0),
                                      LONGREAL)), REAL);
  ELSE
    RETURN 1.0 ;
  END;
END Geom_Mittel;


(*--------------------- Hauptprogramm ----------------------------*)

PROCEDURE SchuelerEingabe() : Schueler =
(* Eingabe eines Schuelers durch Eingabe der einzelnen Werte *)

VAR schueler : Schueler;
  BEGIN
  SIO.PutLine("Bitte den Namen des Schuelers eingeben: ");
  schueler.Name := SIO.GetLine();
  SIO.PutLine("Bitte den Vornamen des Schuelers eingeben: ");
  schueler.Vorname := SIO.GetLine();
  SIO.PutLine("Bitte das Alter des Schuelers eingeben: ");
  schueler.Alter := SIO.GetReal();
  SIO.PutLine("Bitte die Note des Schuelers eingeben: ");
  schueler.Note := SIO.GetReal();
  RETURN schueler;
END SchuelerEingabe;

(* Menue *)

BEGIN
  wahl := 0;
  WHILE wahl # 10 DO
    SIO.PutLine("******************** Menue ********************");
    SIO.PutLine("Waehlen Sie einen Menuepunkt:");
    SIO.Nl();
    SIO.PutLine("Schueler zur Liste hinzufuegen (1)");
    SIO.Nl();
    SIO.PutLine("Schuelerliste ausgeben (2)");
    SIO.Nl();
    SIO.PutLine("Schueler ueber einem einzugebenden Alter ausgeben
                 (3)");
    SIO.Nl();
    SIO.PutLine("Schueler schlechter als eine einzugebende Note
```

```
            ausgeben (4)");
SIO.Nl();
SIO.PutLine("Schueler besser als eine einzugebende Note ausgeben
            (5)," );
SIO.Nl();
SIO.PutLine("Schueler ueber einem einzugebenden Alter und
            schlechter");
SIO.PutLine("schlechter als eine einzugebenden Note ausgeben
            (6)");
SIO.Nl();
SIO.PutLine("Schueler ueber einem einzugebenden Alter und
            besser");
SIO.PutLine("als eine einzugebende Note ausgeben (7)");
SIO.Nl();
SIO.PutLine("Arithmetisches Mittel der Noten berechnen (8)" );
SIO.Nl();
SIO.PutLine("Geometrisches Mittel der Noten berechnen (9)" );
SIO.Nl();
SIO.PutLine("Programm beenden (10)");
SIO.PutLine("****************** Ende Menue ******************");
SIO.Nl();
SIO.PutText("Wahl: ");
wahl := SIO.GetInt();
dummy := SIO.GetLine();
IF wahl = 1 THEN
  schueler := SchuelerEingabe();
  Einfuegen(schuelerListe, schueler);
ELSIF wahl = 2 THEN
  Ausgeben(schuelerListe);
ELSIF wahl = 3 THEN
  sortiert := NIL;
  SIO.PutLine("Bitte die Altersgrenze eingeben: ");
  limitAlter := SIO.GetReal();
  dummy := SIO.GetLine();
  SIO.Nl();
  SIO.PutLine("Schueler ueber ");
  SIO.PutReal(limitAlter);
  SIO.PutLine(" (aufsteigend sortiert nach Alter):");
  SIO.Nl();
  FilterAlter(schuelerListe, limitAlter, sortiert);
  Ausgeben(sortiert);

 ELSIF wahl = 4 THEN
  sortiert := NIL;
  SIO.PutLine("Bitte die Notengrenze eingeben: ");
  limitNote := SIO.GetReal();
  dummy := SIO.GetLine();
  SIO.Nl();
  SIO.PutText("Schueler mit Note ueber ");
  SIO.PutReal(limitNote);
  SIO.PutLine(" (sortiert nach Note):");
  SIO.Nl();
  FilterNote_schlechter_spaeter(schuelerListe, limitNote,
                                sortiert);
  Ausgeben(sortiert);

ELSIF wahl = 5 THEN
  sortiert := NIL;
  SIO.PutLine("Bitte die Notengrenze eingeben: ");
  limitNote := SIO.GetReal();
```

```
    dummy := SIO.GetLine();
    SIO.Nl();
    SIO.PutText("Schueler mit Note unter ");
    SIO.PutReal(limitNote);
    SIO.PutLine(" (sortiert nach Note):");
    SIO.Nl();
    FilterNote_besser_spaeter(schuelerListe, limitNote, sortiert);
    Ausgeben(sortiert);

  ELSIF wahl = 6 THEN
    sortiert := NIL;
    SIO.PutLine("Bitte die Altersgrenze eingeben: ");
    limitAlter := SIO.GetReal();
    dummy := SIO.GetLine();
    SIO.PutLine("Bitte die Notengrenze eingeben: ");
    limitNote := SIO.GetReal();
    dummy := SIO.GetLine();
    SIO.Nl();
    SIO.PutText("Schueler ueber ");
    SIO.PutReal(limitAlter);
    SIO.PutLine(" und Note ueber ");
    SIO.PutReal(limitNote);
    SIO.PutLine(" (sortiert nach Note):");
    SIO.Nl();
    Filtern_schlechter_spaeter(schuelerListe, limitAlter,
                               limitNote, sortiert);
    Ausgeben(sortiert);

  ELSIF wahl = 7 THEN
    sortiert := NIL;
    SIO.PutLine("Bitte die Altersgrenze eingeben: ");
    limitAlter := SIO.GetReal();
    dummy := SIO.GetLine();
    SIO.PutLine("Bitte die Notengrenze eingeben: ");
    limitNote := SIO.GetReal();
    dummy := SIO.GetLine();
    SIO.Nl();
    SIO.PutText("Schueler ueber ");
    SIO.PutReal(limitAlter);
    SIO.PutLine(" und Note unter ");
    SIO.PutReal(limitNote);
    SIO.PutLine(" (sortiert nach Note):");
    SIO.Nl();
    Filtern_besser_spaeter(schuelerListe, limitAlter, limitNote,
                           sortiert);
    Ausgeben(sortiert);

   ELSIF wahl = 8 THEN
     SIO.Nl();
     SIO.Nl();
     IF schuelerListe = NIL THEN
       SIO.PutLine("Die Liste ist leer!");
     ELSE
       SIO.PutText("Arithmetisches Mittel der Noten: ");
       SIO.PutReal(Arithm_Mittel(schuelerListe));
     END;
     SIO.Nl();
    dummy := SIO.GetLine();
   ELSIF wahl = 9 THEN
     SIO.Nl();
```

```
      SIO.Nl();
      IF schuelerListe = NIL THEN
        SIO.PutLine("Die Liste ist leer!");
      ELSE
        SIO.PutText("Geometrisches Mittel der Noten: ");
        SIO.PutReal(Geom_Mittel(schuelerListe));
      END;
      SIO.Nl();
      dummy := SIO.GetLine();
   ELSIF wahl # 10 THEN
      SIO.PutLine("Unzulaessige Eingabe! Bitte neu waehlen!");
      SIO.Nl();
    END;
  END;
SIO.PutLine("Programmende!");
END Schuelerliste.
```

Probelauf des Programms:

```
************************* Menue *************************
Waehlen Sie einen Menuepunkt:

Schueler zur Liste hinzufuegen (1)

Schuelerliste ausgeben (2)

Schueler ueber einem einzugebenden Alter ausgeben (3)

Schueler schlechter als eine einzugebende Note ausgeben (4)

Schueler besser als eine einzugebende Note ausgeben (5),

Schueler ueber einem einzugebenden Alter und schlechter
schlechter als eine einzugebenden Note ausgeben (6)

Schueler ueber einem einzugebenden Alter und besser
als eine einzugebende Note ausgeben (7)

Arithmetisches Mittel der Noten berechnen (8)

Geometrisches Mittel der Noten berechnen (9)

Programm beenden (10)
************************* Ende Menue *************************

Wahl: 1
Bitte den Namen des Schuelers eingeben:
Schlechterschueler
Bitte den Vornamen des Schuelers eingeben:
Emil
Bitte das Alter des Schuelers eingeben:
20
Bitte die Note des Schuelers eingeben:
5.0

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1
```

```
Bitte den Namen des Schuelers eingeben:
Jungerschueler
Bitte den Vornamen des Schuelers eingeben:
Daniel
Bitte das Alter des Schuelers eingeben:
15
Bitte die Note des Schuelers eingeben:
2.0

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1
Bitte den Namen des Schuelers eingeben:
Alterschueler
Bitte den Vornamen des Schuelers eingeben:
Christina
Bitte das Alter des Schuelers eingeben:
25
Bitte die Note des Schuelers eingeben:
4.0

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1

Bitte den Namen des Schuelers eingeben:
Musterschueler
Bitte den Vornamen des Schuelers eingeben:
Berta
Bitte das Alter des Schuelers eingeben:
17
Bitte die Note des Schuelers eingeben:
1.0

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 1

Bitte den Namen des Schuelers eingeben:
Schueler
Bitte den Vornamen des Schuelers eingeben:
Anton
Bitte das Alter des Schuelers eingeben:
18
Bitte die Note des Schuelers eingeben:
3.0

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 2
Name: Schueler
Vorname: Anton
```

```
Alter: 18
Note: 3

-------------------------------------------------

Name: Musterschueler
Vorname: Berta
Alter: 17
Note: 1

-------------------------------------------------

Name: Alterschueler
Vorname: Christina
Alter: 25
Note: 4

-------------------------------------------------

Name: Jungerschueler
Vorname: Daniel
Alter: 15
Note: 2

-------------------------------------------------

Name: Schlechterschueler
Vorname: Emil
Alter: 20
Note: 5

-------------------------------------------------

************************** Menue **************************
...
************************** Ende Menue **************************

Wahl: 3
Bitte die Altersgrenze eingeben:
17

Schueler ueber
17 (aufsteigend sortiert nach Alter):

Name: Schueler
Vorname: Anton
Alter: 18
Note: 3

-------------------------------------------------

Name: Schlechterschueler
Vorname: Emil
Alter: 20
Note: 5

-------------------------------------------------

Name: Alterschueler
Vorname: Christina
```

```
Alter: 25
Note: 4

-----------------------------------------------

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 4
Bitte die Notengrenze eingeben:
2.5

Schueler mit Note ueber 2.5 (sortiert nach Note):

Name: Schueler
Vorname: Anton
Alter: 18
Note: 3

-----------------------------------------------

Name: Alterschueler
Vorname: Christina
Alter: 25
Note: 4

-----------------------------------------------

Name: Schlechterschueler
Vorname: Emil
Alter: 20
Note: 5

-----------------------------------------------

************************* Menue *************************
...
************************* Ende Menue *************************
Wahl: 5
Bitte die Notengrenze eingeben:
4

Schueler mit Note unter 4 (sortiert nach Note):

Name: Schueler
Vorname: Anton
Alter: 18
Note: 3

-----------------------------------------------

Name: Jungerschueler
Vorname: Daniel
Alter: 15
Note: 2

-----------------------------------------------

Name: Musterschueler
```

```
Vorname: Berta
Alter: 17
Note: 1

------------------------------------------------

************************* Menue **************************
...
************************* Ende Menue **************************

Wahl: 6
Bitte die Altersgrenze eingeben:
16
Bitte die Notengrenze eingeben:
3

Schueler ueber 16 und Note ueber
3 (sortiert nach Note):

Name: Alterschueler
Vorname: Christina
Alter: 25
Note: 4

------------------------------------------------

Name: Schlechterschueler
Vorname: Emil
Alter: 20
Note: 5

------------------------------------------------

************************* Menue **************************
...
************************* Ende Menue **************************

Wahl: 7
Bitte die Altersgrenze eingeben:
15
Bitte die Notengrenze eingeben:
5

Schueler ueber 15 und Note unter
5 (sortiert nach Note):

Name: Alterschueler
Vorname: Christina
Alter: 25
Note: 4

------------------------------------------------

Name: Schueler
Vorname: Anton
Alter: 18
Note: 3

------------------------------------------------
```

```
Name: Musterschueler
Vorname: Berta
Alter: 17
Note: 1

-----------------------------------------------

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 8

Arithmetisches Mittel der Noten: 3

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 9

Geometrisches Mittel der Noten: 2.6051712

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 2
Name: Schueler
Vorname: Anton
Alter: 18
Note: 3

-----------------------------------------------

Name: Musterschueler
Vorname: Berta
Alter: 17
Note: 1

-----------------------------------------------

Name: Alterschueler
Vorname: Christina
Alter: 25
Note: 4

-----------------------------------------------

Name: Jungerschueler
Vorname: Daniel
Alter: 15
Note: 2

-----------------------------------------------

Name: Schlechterschueler
Vorname: Emil
Alter: 20
Note: 5
```

```
------------------------------------------------

************************* Menue *************************
...
************************* Ende Menue *************************

Wahl: 10
Programmende!
```

Programmierung **WS 2000/2001**

Prof. H. Lichter
M. Schnizler

Übung 9
Musterlösung

Aufgabe 9.1

Aufwand der Funktion EMPotenz:

$A_{EMP}(n) = n * 1 = n$

Begründung: Die FOR-Schleife wird genau n-mal durchlaufen, wobei jedes Mal eine Multiplikation durchgeführt wird.

Aufwand der Funktion SQPotenz:

Zunächst Berechnung einiger Beispielwerte:

$A_{SQP}(0) = 0$ => Keine Multiplikation!
$A_{SQP}(1) = A_{SQP}(0) + 1$ => Rekursion mit 0 + 1 Multiplikation

$A_{SQP}(2) = A_{SQP}(2 \text{ DIV } 2) + 1 = A_{SQP}(1) + 1 = 2$
$A_{SQP}(3) = A_{SQP}(3 - 1) + 1 = A_{SQP}(2) + 1 = A_{SQP}(2 \text{ DIV } 2) + 1 + 1 = A_{SQP}(1) + 2 = 3$
$A_{SQP}(4) = A_{SQP}(4 \text{ DIV } 2) + 1 = A_{SQP}(2) + 1 = A_{SQP}(2 \text{ DIV } 2) + 1 + 1 = A_{SQP}(1) + 2 = 3$
$A_{SQP}(5) = A_{SQP}(4) + 1 = A_{SQP}(2) + 2 = A_{SQP}(1) + 3 = 4$
$A_{SQP}(6) = A_{SQP}(3) + 1 = A_{SQP}(2) + 2 = A_{SQP}(1) + 3 = 4$
$A_{SQP}(7) = A_{SQP}(6) + 1 = A_{SQP}(3) + 2 = A_{SQP}(2) + 3 = A_{SQP}(1) + 4 = 5$
$A_{SQP}(8) = A_{SQP}(4) + 1 = A_{SQP}(2) + 2 = A_{SQP}(1) + 3 = 4$

Im folgenden sei $ld\ n = \log_2 n$ der ganzzahlige Wert des Logarithmus zur Basis 2.

Untere Schranke:
Betrachtet man die Beispiele, so zeigt sich, dass der Aufwand für n = 8 den günstigsten Fall darstellt. Der Exponent n ist eine ganzzahlige Potenz von 2, nämlich $n = 2^3 = 8$ und da der Exponent in jedem Rekursionsschritt halbiert wird, ergibt sich folgender Aufwand:

$A_{SQP}(n) = ld\ n + 1$ (für $n = 2^m$ und $m \geq 0$)

Obere Schranke:
Im ungünstigsten Fall ist der Exponent ungerade und es entsteht bei jeder darauffolgenden Division durch 2 wiederum eine ungerade Zahl. Das ist beispielsweise der Fall bei n = 7, zu Beginn und nach jeder Division ist eine zusätzliche Multiplikation für die Rekursion mit dem um eins verringerten Wert notwendig:

$A_{SQP}(n) = 2 * ld\ n + 1$ (für $n = 2^m - 1$ und $m \geq 0$)

Vergleich der Ergebnisse:

$A_{EMP}(1024) = 1024$ $A_{SQP}(1024) = ld\ 1024 + 1 = 10 + 1 = 11$ Faktor: $1024 / 11 \approx 93$
$A_{EMP}(1023) = 1023$ $A_{SQP}(1023) = 2 * ld\ 1023 + 1 = 18 + 1 = 19$ Faktor: $1023 / 19 \approx 54$

Aufgabe 9.2

```
MODULE TerminDatei EXPORTS Main;

(* Dieses Programm realisiert eine simple Terminverwaltung, die
   zuvor eingegebene Termine sortiert ausgibt. Die Termine koennen
   ausserdem in einer Datei gespeichert und von dort wieder gelesen
   werden.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 23.11.00
   Letzte Aenderung: 19.01.01 *)

IMPORT SIO;    (* Importiere notwendige Ein-/Ausgabeoperationen *)
IMPORT IO;     (* Importiere Operationen um Dateien zu oeffnen*)
IMPORT Rd, Wr; (* Importiere Op. um Dateien zu lesen/schreiben *)

CONST MAXTERMIN = 10;         (* Maximale Anzahl der Termine *)
      KALENDER  = "Kalender"; (* Name der Ausgabedatei *)

TYPE (* Unterbereichtstypen und Typ Uhrzeit fuer die Zeitangabe *)

     Stunden        = [0..23];

     Minuten        = [0..59];

     Uhrzeit        = RECORD
                        stunde: Stunden;
                        minute: Minuten;
                      END;


     (* Typ Termin fuer einzelnen Termin mit Uhrzeit/Beschreibung *)

     Termin         = RECORD
                        beschreibung: TEXT;
                        zeit        : Uhrzeit;
                      END;


     (* Typ Terminkalender fuer maximal MAXTERMIN Termine *)

     Terminkalender = ARRAY [1..MAXTERMIN] OF Termin;


(* ... unveränderte Prozeduren und Funktionen aus Aufg. 6.2 werden
       im weiteren nicht aufgeführt ... *)
```

```
(* Prozeduren fuer das Schreiben und Lesen der Termindatei *)

PROCEDURE TerminLesen(datei: Rd.T): Termin =
(* Liest die Daten des Termins aus einer gegebenen Datei *)
VAR ergTermin   : Termin;
    trennzeichen: CHAR;

BEGIN
  ergTermin.zeit.stunde := SIO.GetInt(datei);
  trennzeichen := SIO.GetChar(datei);
  ergTermin.zeit.minute := SIO.GetInt(datei);
  trennzeichen := SIO.GetChar(datei);
  ergTermin.beschreibung := SIO.GetLine(datei);

  RETURN ergTermin;
END TerminLesen;


PROCEDURE TerminSchreiben(datei: Wr.T; termin: Termin) =
(* Schreibt die Daten des Termins in die gegebene Datei im Format:
   Stunde:MinuteBeschreibungRETURN *)
BEGIN
  SIO.PutInt(termin.zeit.stunde, 10, datei);
  SIO.PutChar(':', datei);  (* Als Trennzeichen *)
  SIO.PutInt(termin.zeit.minute, 10, datei);
  SIO.PutChar(' ', datei);         (* Als Trennzeichen *)
  SIO.PutLine(termin.beschreibung, datei);
END TerminSchreiben;


PROCEDURE TerminkalenderLesen(datei: Rd.T; VAR termine:
Terminkalender; VAR maxIndex: [0..MAXTERMIN]) =
(* Liest alle (maximal MAXTERMIN viele) Termine eines
Terminkalenders aus der gegebenen Datei.
   Der gefüllte Terminkalender und der maximal belegte Index werden
zurueckgegeben. *)
VAR i: INTEGER := 0;
BEGIN

  WHILE NOT SIO.EOF(datei) AND (i <= MAXTERMIN) DO
    INC(i);
    termine[i] := TerminLesen(datei);
  END;

  maxIndex := i;
END TerminkalenderLesen;


PROCEDURE TerminkalenderSchreiben(datei: Wr.T; termine:
Terminkalender; maxIndex: INTEGER) =
(* Schreibt den bis maxIndex gefuellten Terminkalender in die
gegebene Datei *)
BEGIN
  FOR i := FIRST(termine) TO maxIndex DO
    TerminSchreiben(datei, termine[i]);
  END;
END TerminkalenderSchreiben;
```

```
VAR tKalender  : Terminkalender;
    terminIndex: [0..MAXTERMIN] := 0;

    (* Variablen fuer zum Schreiben bzw. Lesen geoeffnete Datei *)
    schreibDatei: Wr.T;
    leseDatei   : Rd.T;

    (* Notwendig fuer die Benutzerinteraktion *)
    auswahl    : CHAR;
    restzeile  : TEXT;

BEGIN
  (* Lesen der Terminkalenderdatei *)
  leseDatei := IO.OpenRead(KALENDER);
  IF (leseDatei # NIL) THEN
    TerminkalenderLesen(leseDatei, tKalender, terminIndex);
    Rd.Close(leseDatei);
  END;

  (* Benutzerinteraktion mittels REPEAT-Schleife *)
  REPEAT

    (* Auswahlmenue ausgeben *)
    SIO.Nl();
    SIO.PutLine("***  Terminkalender  ***");
    SIO.PutLine("(e) Neuen Termin eingeben");
    SIO.PutLine("(+) Aufsteigend sortiert ausgeben");
    SIO.PutLine("(-) Absteigend sortiert ausgeben");
    SIO.PutLine("(q) Beendet das Programm");
    SIO.Nl();

    (* Eingabe der ausgewaehlten Operation *)
    SIO.PutText("Auswahl: ");
    auswahl   := SIO.GetChar();
    restzeile := SIO.GetLine();
    SIO.Nl();

    (* Ausgewaehlte Operation ausfuehren *)
    CASE auswahl OF
       'e', 'E' => IF terminIndex < MAXTERMIN THEN
                     (* Solange noch Platz im Terminkalender *)
                     INC(terminIndex);
                     tKalender[terminIndex] :=TerminEingeben();

                     (* Terminkalender sofort sortieren *)
                     TerminkalenderSortieren(tKalender,terminIndex);
                   ELSE
                     SIO.PutLine("Terminkalender leider voll!");
                   END;

    |  '+'      => (* Terminkalender aufsteigend ausgeben *)
                   TerminkalenderAusgeben(tKalender, terminIndex,
                                          TRUE);

    |  '-'      => (* Terminkalender absteigend ausgeben *)
                   TerminkalenderAusgeben(tKalender, terminIndex,
                                          FALSE);
```

```
    |  'q', 'Q' => (* Kalenderdaten in Datei schreiben *)
                   schreibDatei := IO.OpenWrite(KALENDER);

                   IF (schreibDatei # NIL) THEN
                     TerminkalenderSchreiben(schreibDatei,tKalender,
                                             terminIndex);
                     Wr.Close(schreibDatei);
                   ELSE
                     SIO.PutLine("Kann Datei nicht schreiben!");
                   END;

                   SIO.PutLine("Programmende!");
    ELSE
      SIO.PutLine("Ungueltige Eingabe!");
    END;
  UNTIL (auswahl = 'q') OR (auswahl = 'Q');
END TerminDatei.
```

Probelauf des Programms, um drei Termine einzugeben:

```
***  Terminkalender  ***
(e) Neuen Termin eingeben
(+) Aufsteigend sortiert ausgeben
(-) Absteigend sortiert ausgeben
(q) Beendet das Programm

Auswahl: e

Beschreibung des Termins: Frühstücken
Uhrzeit (Stunden): 6
Uhrzeit (Minuten): 00

***  Terminkalender  ***
...

Auswahl: e

Beschreibung des Termins: Abendessen
Uhrzeit (Stunden): 20
Uhrzeit (Minuten): 30

***  Terminkalender  ***
...

Auswahl: e

Beschreibung des Termins: Mittagessen
Uhrzeit (Stunden): 13
Uhrzeit (Minuten): 00
```

Inhalt der erzeugten Beispieldatei:

```
6:0Fr hst cken
13:0Mittagessen
20:30Abendessen
```

Aufgabe 9.3

a) Geeignete Äquivalenzklassen zu den gegebenen Eingabebedingungen:

Eingabebedingung	Äquivalenzklassen	
	Gültig	Ungültig
Familienname notwendig	ein oder mehr Zeichen (1)	leerer Text (2)
Rufname notwendig	ein oder mehr Zeichen (3)	leerer Text (4)
maximal fünf Vornamen	keiner oder maximal vier weitere Vornamen (5)	fünf oder mehr weitere Vornamen (6)
nach greg. Kalender gültiges Geburtsdatum notwendig	gültiges Datum von 1800 - 3000 außer Schalttage (7), Schalttag von 1800 - 3000 (8)	ungültiges Datum von 1800 bis 3000 (9), ungültiges Datum vor 1800 oder nach 3000 (10), gültiges Datum vor 1800 (11), gültiges Datum nach 3000 (12)
Geburtsort notwendig	ein oder mehr Zeichen (13)	leerer Text (14)
Geschlecht notwendig	{männlich, weiblich} (15)	sonstige Eingabe (16)
Strassenangabe notwendig	ein oder mehr Zeichen (17)	leerer Text (18)
Hausnummer von 1 bis 1000	Wert von 1 bis 1000 (19)	Wert < 1 (20), Wert > 1000 (21)
Adresszusatz freiwillig	beliebige Eingabe (22)	
Postleitzahl von 0 bis 99999	Wert von 0 bis 99999 (23)	Wert < 0 (24), Wert > 99999 (25)
Ortsangabe notwendig	ein oder mehr Zeichen (26)	leerer Text (27)
Familienstand notwendig	{verh., ledig, gesch., verw.}	sonstige Eingabe (28)
falls verheiratet, Name des Ehepartners erforderlich	{verh.} (29), {ledig, gesch., verw.} (30)	sonstige Eingabe (28)
falls verheiratet, Familien-name des EP notwendig	ein oder mehr Zeichen (31)	leerer Text (32)
falls verheiratet, Rufname des EP notwendig	ein oder mehr Zeichen (33)	leerer Text (34)
falls verheiratet, max. fünf Vornamen des EP	kein oder maximal vier weitere Vornamen (35)	fünf oder mehr weitere Vornamen (36)
nach greg. Kalender gültiges Anmeldedatum notwendig	gültiges Datum von 1800 - 3000 außer Schalttage (37), Schalttag von 1800 - 3000 (38)	ungültiges Datum von 1800 bis 3000 (39), ungültiges Datum vor 1800 oder nach 3000 (40), gültiges Datum vor 1800 (41), gültiges Datum nach 3000 (42)

Testfälle, welche die oben angegebenen Äquivalenzklassen überdecken:

Testfallnummer	1	2	3	4	5	6	7	8	9
Familienname	M	Meier	Meier		Meier	Testfall	Meier	Meier	Meier
Rufname	W	Wilhelm	Wilhelm	Wilhelm		nicht	Wilhelm	Wilhelm	Wilhelm
2. Vorname		Albert	Albert	Albert	Albert	möglich!	Albert	Albert	Albert
3. Vorname		Friedrich							
4. Vorname		Franz							
5. Vorname		Ludwig							
Geb.datum	01.01.1800	31.12.3000	29.02.1980	20.01.2000	20.01.2000		30.02.1980	32.03.1799	20.03.1799
Geburtsort	B	Bonn	Bonn	Bonn	Bonn		Bonn	Bonn	Bonn
Geschlecht	männlich	männlich	männlich	männlich	männlich		männlich	männlich	männlich
Strasse	S	Spreeweg	Spreeweg	Spreeweg	Spreeweg		Spreeweg	Spreeweg	Spreeweg
Hausnummer	1	1000	500	500	500		500	500	500
Adresszusatz		A	A	A	A		A	A	A
Postleitzahl	0	99999	50000	50000	50000		50000	50000	50000
Ort	B	Bonn	Bonn	Bonn	Bonn		Bonn	Bonn	Bonn
Familienstand	verheiratet	verheiratet	ledig	ledig	ledig		ledig	ledig	ledig
Familienname EP	W	Walter							
Rufname EP	O	Olga							
2. Vorname EP		Erika							
3. Vorname EP		Helga							
4. Vorname EP		Erna							
5. Vorname EP		Lora							
Anmeldedatum	01.01.1800	31.12.3000	29.02.1904	20.03.1880	20.03.1880		20.03.1880	20.03.1980	20.03.1980
Sollresultat	OK	OK	OK	Familien-name fehlt!	Rufname fehlt!	Zuviele Vornamen!	Ungültiges Geb.datum!	Ungültiges Geb.datum!	Geb.datum zu früh!

Testfallnummer	10	11	12	13	14	15	16	17	18
Familienname	Meier	Meier	Meier	Meier	Meier	Meier	Meier	Meier	Meier
Rufname	Wilhelm	Wilhelm	Wilhelm	Wilhelm	Wilhelm	Wilhelm	Wilhelm	Wilhelm	Wilhelm
2. Vorname	Albert	Albert	Albert	Albert	Albert	Albert	Albert	Albert	Albert
3. Vorname									
4. Vorname									
5. Vorname									
Geb.datum	01.01.3001	20.03.1988	20.03.1988	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980
Geburtsort	Bonn		Bonn	Bonn	Bonn	Bonn	Bonn	Bonn	Bonn
Geschlecht	männlich	männlich	xy	männlich	männlich	männlich	männlich	männlich	männlich
Strasse	Spreeweg	Spreeweg	Spreeweg		Spreeweg	Spreeweg	Spreeweg	Spreeweg	Spreeweg
Hausnummer	500	500	500	500	0	1001	500	500	500
Adresszusatz	A	A	A	A	A	A	A	A	A
Postleitzahl	50000	50000	50000	50000	50000	50000	-1	100000	50000
Ort	Bonn	Bonn	Bonn	Bonn	Bonn	Bonn	Bonn	Bonn	
Familienstand	ledig	ledig	ledig	ledig	ledig	ledig	ledig	ledig	ledig
Familienname EP									
Rufname EP									
2. Vorname EP									
3. Vorname EP									
4. Vorname EP									
5. Vorname EP									
Anmeldedatum	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980	20.03.1980
Sollresultat	Geb.datum zu spät!	Geburtsort fehlt!	Geschlecht falsch!	Strasse fehlt!	Nummer falsch!	Nummer falsch!	PLZ falsch!	PLZ falsch!	Wohnort fehlt!

Testfallnummer	19	20	21	22	23	24	25	26	27 (Zusatz)
Familienname	Meier	Meier	Meier	Testfall	Meier	Meier	Meier	Meier	Meier
Rufname	Wilhelm	Wilhelm	Wilhelm	nicht	Wilhelm	Wilhelm	Wilhelm	Wilhelm	Wilhelm
2. Vorname	Albert	Albert	Albert	möglich!	Albert	Albert	Albert	Albert	Albert
3. Vorname									
4. Vorname									
5. Vorname									
Geb.datum	20.03.1988	20.03.1980	20.03.1988		20.03.1980	20.03.1980	20.03.2000	20.03.2000	20.03.2000
Geburtsort	Bonn	Bonn	Bonn		Bonn	Bonn	Bonn	Bonn	Bonn
Geschlecht	männlich	männlich	männlich		männlich	männlich	männlich	männlich	männlich
Strasse	Spreeweg	Spreeweg	Spreeweg		Spreeweg	Spreeweg	Spreeweg	Spreeweg	Spreeweg
Hausnummer	500	500	500		500	500	500	500	500
Adresszusatz	A	A	A		A	A	A	A	A
Postleitzahl	50000	50000	50000		50000	50000	50000	50000	50000
Ort	Bonn	Bonn	Bonn		Bonn	Bonn	Bonn	Bonn	Bonn
Familienstand	unbekannt	verheiratet	verheiratet		ledig	ledig	ledig	ledig	ledig
Familienname EP			Walter						
Rufname EP		Olga							
2. Vorname EP		Erika	Erika						
3. Vorname EP									
4. Vorname EP									
5. Vorname EP									
Anmeldedatum	20.03.1980	20.03.1980	20.03.1980		4.13.1960	43.12.3001	31.12.1799	01.01.3001	29.02.1983
Sollresultat	Familienstand falsch!	Fam.name EP fehlt!	Rufname EP fehlt!		Ungültiges Anmdatum!	Ungültiges Anmdatum!	Ungültiges Anmdatum!	Ungültiges Anmdatum!	Ungültiges Anmdatum!

Tabelle der Überdeckung Testfall/Äquivalenzklassen:

Testfall	Äquivalenzklassen																																									
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
1	U		U		U		U						U		X		U		U			X	U			U			X		U		U		U		U					
2	X		X		O		O						X		X		X		O			X	O			X			X		X		X		O		O					
3	X		X		X			X					X		X		X		X			X	X			X				X								X				
4		X	X		X		X						X		X		X		X			X	X			X				X							X					
5	X			X	X		X						X		X		X		X			X	X			X				X							X					
6	X		X			X	X						X		X		X		X			X	X			X				X							X					
7	X		X		X				X				X		X		X		X			X	X			X				X							X					
8	X		X		X					X			X		X		X		X			X	X			X				X							X					
9	X		X		X						X		X		X		X		X			X	X			X				X							X					
10	X		X		X							X	X		X		X		X			X	X			X				X							X					
11	X		X		X		X							X	X		X		X			X	X			X				X							X					
12	X		X		X		X						X			X	X		X			X	X			X				X							X					
13	X		X		X		X						X		X			X	X			X	X			X				X							X					
14	X		X		X		X						X		X		X			X		X	X			X				X							X					
15	X		X		X		X						X		X		X				X	X	X			X				X							X					
16	X		X		X		X						X		X		X		X			X		X		X				X							X					
17	X		X		X		X						X		X		X		X			X			X	X				X							X					
18	X		X		X		X						X		X		X		X			X	X				X			X							X					
19	X		X		X		X						X		X		X		X			X	X			X		X									X					
20	X		X		X		X						X		X		X		X			X	X			X			X			X	X		X		X					
21	X		X		X		X						X		X		X		X			X	X			X			X		X			X	X		X					
22	X		X		X		X						X		X		X		X			X	X			X			X		X		X			X	X					
23	X		X		X		X						X		X		X		X			X	X			X				X									X			
24	X		X		X		X						X		X		X		X			X	X			X				X										X		
25	X		X		X		X						X		X		X		X			X	X			X				X											X	
26	X		X		X		X						X		X		X		X			X	X			X				X												X

U = Untergrenze Äquivalenzklasse
O = Obergrenze Äquivalenzklasse
X = beliebiger Wert aus Äquivalenzklasse

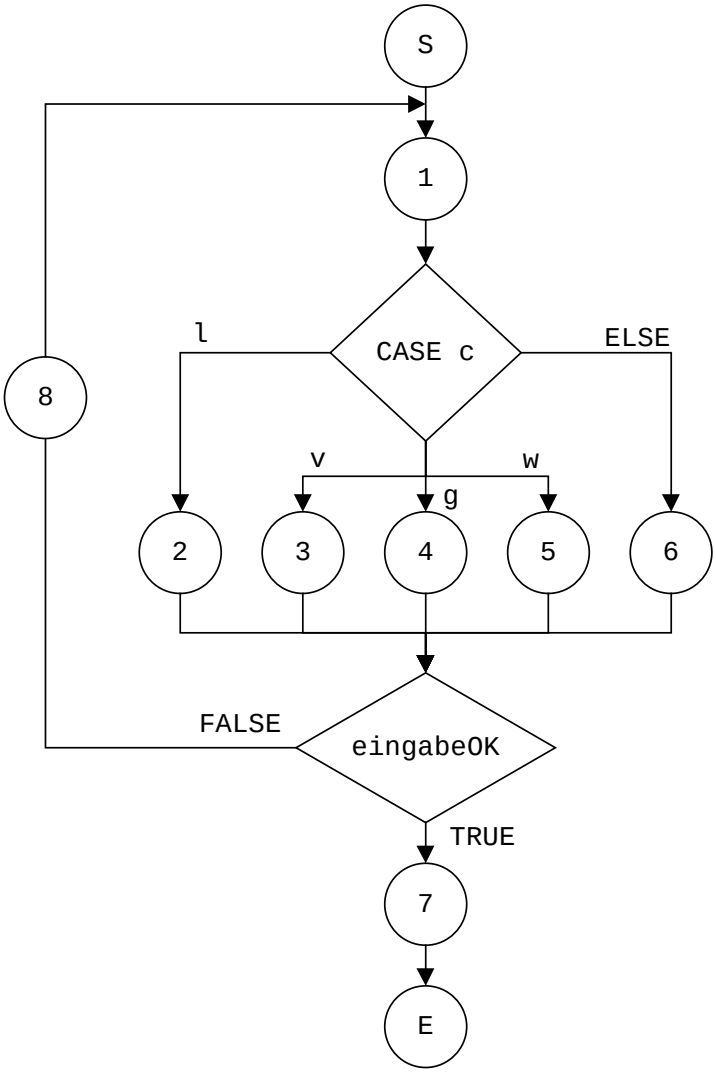
HINWEIS: Bei der Bestimmung der Äquivalenzklassen und der Grenzwertanalyse werden keine Abhängigkeiten zwischen den Eingaben berücksichtigt, sondern nur die einzelnen Datenwerte betrachtet. Beispielsweise wird bei der Testfallauswahl nicht erzwungen, dass das Anmeldedatum später als das Geburtsdatum liegt. Auch lassen sich durch eine andere Wahl der Äquivalenzklassen (vor allem bezüglich des Datums) noch wesentlich mehr Testfälle ermitteln, die mit Sicherheit weitere Fehler finden. Dies erfordert deutlich mehr Aufwand, der sich allerdings lohnt, wie sich allein an dem zusätzlichen Testfall 27 zeigt.

b) Festgestellte Abweichungen zwischen Soll- und Ist-Resultat:

Testfall	Abweichung
4	Keine Meldung, wenn Familienname fehlt!
5	Keine Meldung, wenn Rufname fehlt!
6	Ungültiges Datum wird akzeptiert!
11	Keine Meldung, wenn Geburtsort fehlt!
13	Keine Meldung, wenn Strasse fehlt!
18	Keine Meldung, wenn Wohnort fehlt!
20	Keine Meldung, wenn Familienname Ehepartner fehlt!
21	Keine Meldung, wenn Rufname Ehepartner fehlt!
27	Falscher Schalttag wird akzeptiert!

Erzielte Zweigüberdeckung: 89,6 % (beispielsweise wird in keinem Testfall als Familienstand "geschieden" oder "verwitwet" verwendet)

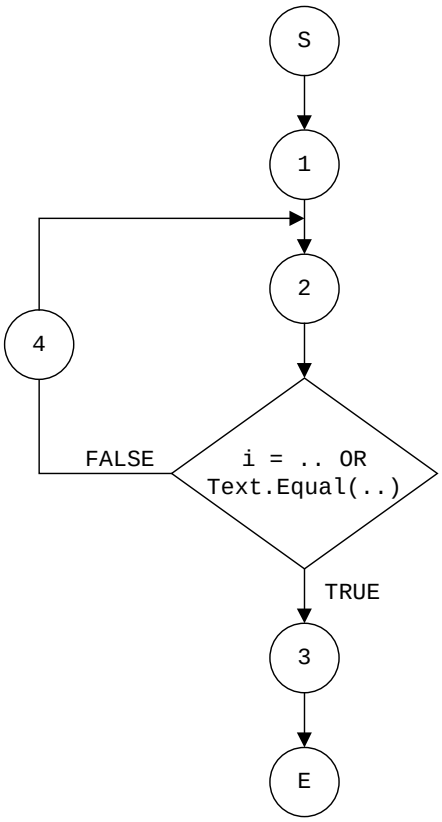
c) Flussdiagramm für Prozedur `FamilienstandEingeben()`:



Testfälle für Zweigüberdeckung:

Testfall	Eingabe für c	Soll-Rückgabewert	Überdeckte Zweige
1	'l'	Familienstand.ledig	1, 2, 7
2	'v'	Familienstand.verheiratet	1, 3, 7
3	'g'	Familienstand.geschieden	1, 4, 7
4	'w'	Familienstand.verwitwet	1, 5, 7
5	'x' 'l'	Familienstand.ledig	1, 6, 8, 1, 2, 7

c) Flussdiagramm für Prozedur `PersonEingeben()`:



Testfälle für eingeschränkte Pfadüberdeckung:

Testfall	Eingabe für Person	Soll-Rückgabewert für Person	Überdeckter Pfad
1	"Müller" "Werner" ""	Müller, Werner, , , ,	Keine Iteration
2	"Müller" "Werner" "Albert" ""	Müller, Werner, Albert, , ,	1 Iteration
3	"Müller" "Werner" "Albert" "Franz" "Ludwig" "Wilhelm"	Müller, Werner, Albert, Franz, Ludwig, Wilhelm	3 Iterationen

Programmierung **WS 2000/2001**

Prof. H. Lichter
Th. von der Maßen

Übung 10
Musterlösung

Aufgabe 10.1

a)

```
INTERFACE CD;

(* Schnittstelle des abstrakten Datentyps CD.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 02.01.01
*)

IMPORT Track;   (* Importiere den ADT Track *)

TYPE T <: REFANY;

PROCEDURE GetInterpret(cd: T): TEXT;
(* Liefert den Interpreten einer CD *)

PROCEDURE SetInterpret(VAR cd: T; interpret: TEXT);
(* Setzt den Interpreten der angegebenen CD *)

PROCEDURE GetTitle(cd: T): TEXT;
(* Liefert den Titel der übergebenen CD *)

PROCEDURE SetTitle(VAR cd: T; title: TEXT);
(* Setzt den Titel der übergebenen CD *)

PROCEDURE GetTotalTime(cd: T): CARDINAL;
(* Liefert die Gesamtspielzeit der CD in Sekunden *)

PROCEDURE AddTrack(VAR cd: T; track: Track.T);
(* Fügt einen Track der übergebenen CD hinzu *)

PROCEDURE RemoveTrack(VAR cd: T);
(* Löscht den letzten Track der übergebenen CD *)

PROCEDURE GetTrackCount(cd: T): CARDINAL;
(* Liefert die Anzahl der Tracks der übergebenen CD *)

PROCEDURE PrintTracklist(cd: T);
(* Zeigt die Trackliste der CD auf dem Bildschirm an *)

PROCEDURE CreateCD(): T;
(* Erstellt eine neue CD und initalisiert diese *)

END CD.
```

```
INTERFACE Track;

(* Schnittstelle des abstrakten Datentyps Track.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 02.01.01
*)

TYPE T <: REFANY;

PROCEDURE GetName(track: T): TEXT;
(* Liefert den Namen des Tracks *)

PROCEDURE SetName(VAR track: T; name: TEXT);
(* Setzt den Namen des Tracks auf den übergebenen String *)

PROCEDURE GetLength(track: T): CARDINAL;
(* Liefert die Länge des Tracks in Sekunden *)

PROCEDURE SetLength(VAR track: T; length: CARDINAL);
(* Setzt die Länge des übergebenen Tracks in Sekunden *)

PROCEDURE CreateTrack(): T;
(* Erstellt einen neuen Track und initialisiert diesen *)

END Track.
```

```
INTERFACE Diskographie;

(* Schnittstelle des Objektmoduls zur Verwaltung einer
Diskographie.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 02.01.01
*)

IMPORT CD;  (* Importiere den ADT CD *)

PROCEDURE Init();
(* Initialisiert die Diskographie. Diese enthält dann keine CDs
*)

PROCEDURE AddCD(cd: CD.T);
(* Fügt die übergebene CD der Diskographie hinzu *)

PROCEDURE GetCD(index: CARDINAL): CD.T;
(* Liefert die CD der Diskographie mit dem übergebenen Index *)

PROCEDURE RemoveCD(pos: CARDINAL);
(* Löscht die CD an der übergebenen Indexposition *)
```

```
PROCEDURE IsFull(): BOOLEAN;
(* Prüft, ob die Diskographie voll ist *)

PROCEDURE IsEmpty(): BOOLEAN;
(* Prüft, ob die Diskographie leer ist *)

PROCEDURE Print();
(* Listet den Inhalt der Diskographie auf dem Bildschirm *)

PROCEDURE PrintWithTracklist();
(* Listet den Inhalt der Diskographie inklusive der Trackliste
jeder CD auf dem Bildschirm *)

END Diskographie.
```

b)

```
MODULE CD;

(* Implementierung des abstrakten Datentyps CD.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 02.01.01
*)

IMPORT SIO;
IMPORT Track;    (* Importiere den ADT Track *)


(* Die Trackliste wird als lineare Liste realisiert *)
TYPE Tracklist = REF RECORD
                   track: Track.T;
                   next: Tracklist;
                 END;

(* Eine CD besteht aus einem Interpreten, einem Titel und einer
Liste von
Musikstücken *)
     REVEAL T = BRANDED REF RECORD
                  interpret: TEXT;
                  title: TEXT;
                  tracklist: Tracklist;
                END;

PROCEDURE GetInterpret(cd: T): TEXT =
(* Liefert den Interpreten einer CD *)
BEGIN
  RETURN (cd^.interpret);
END GetInterpret;


PROCEDURE SetInterpret(VAR cd: T; interpret: TEXT) =
(* Setzt den Interpreten der angegebenen CD *)
BEGIN
  cd^.interpret := interpret;
END SetInterpret;


PROCEDURE GetTitle(cd: T): TEXT =
(* Liefert den Titel der übergebenen CD *)
BEGIN
  RETURN (cd^.title);
END GetTitle;


PROCEDURE SetTitle(VAR cd: T; title: TEXT) =
(* Setzt den Titel der übergebenen CD *)
BEGIN
  cd^.title := title;
```

```
END SetTitle;


PROCEDURE GetTotalTime(cd: T): CARDINAL =
(* Liefert die Gesamtspielzeit der CD in Sekunden *)
VAR totaltime: CARDINAL;
    tlist : Tracklist;
    t : Track.T;
BEGIN
  tlist := cd^.tracklist;
  totaltime := 0;
  (* Summiere die Längen der einzelnen Tracks der CD *)
  WHILE (tlist # NIL) DO
    t := tlist^.track;
    totaltime := totaltime + Track.GetLength(t);
    tlist := tlist^.next;
  END;
  RETURN totaltime;
END GetTotalTime;

PROCEDURE AddTrack(VAR cd: T; track: Track.T) =
(* Fügt einen Track der übergebenen CD hinzu. Der neue Track wird
an das Ende der Trackliste angehängt. *)
VAR newtrack, actual, prev: Tracklist;
BEGIN
  newtrack := NEW(Tracklist);
  newtrack^.track := track;
  newtrack^.next := NIL;

  (* Falls Trackliste leer *)
  IF (cd^.tracklist = NIL) THEN
    cd^.tracklist := newtrack;
  ELSE
    (* Suche Ende der Trackliste *)
    actual := cd^.tracklist;
    prev := cd^.tracklist;

    WHILE (actual # NIL) DO
      prev := actual;
      actual := actual^.next;
    END;

    prev^.next := newtrack;
  END;
END AddTrack;


PROCEDURE RemoveTrack(VAR cd: T) =
(* Löscht den letzten Track der übergebenen CD *)
VAR actual, prev: Tracklist;
BEGIN
  IF (cd^.tracklist = NIL) THEN
    SIO.PutLine("Trackliste ist leer!");
  ELSIF (cd^.tracklist.next = NIL) THEN
    cd^.tracklist := NIL;
  ELSE
    actual := cd^.tracklist;
    prev := cd^.tracklist;
    WHILE (actual^.next # NIL) DO
      prev := actual;
      actual := actual^.next;
    END;
    prev^.next := NIL;
  END;
END RemoveTrack;


PROCEDURE GetTrackCount(cd: T): CARDINAL =
(* Liefert die Anzahl der Tracks der übergebenen CD *)
VAR count: CARDINAL;
    tl: Tracklist;
BEGIN
  count := 0;
  tl := cd^.tracklist;

  WHILE (tl # NIL) DO
    count := count + 1;
    tl := tl^.next;
  END;
  RETURN count;
END GetTrackCount;


PROCEDURE PrintTracklist(cd: T) =
(* Zeigt die Trackliste der CD mit Namen und Länge der Tracks
auf dem Bildschirm an *)
VAR tl: Tracklist;
    count : CARDINAL;
BEGIN
  tl := cd^.tracklist;
  count := 1;
  WHILE (tl # NIL) DO
    SIO.PutInt(count); SIO.PutText(". ");
    SIO.PutText(Track.GetName(tl^.track) & " Dauer: ");
    SIO.PutInt(Track.GetLength(tl^.track)); SIO.Nl();
    tl := tl^.next;
    count := count + 1;
  END;
END PrintTracklist;


PROCEDURE CreateCD(): T =
(* Erstellt eine neue CD und initalisiert diese *)
VAR newcd: T;
BEGIN
  newcd := NEW(T);
```

```
  newcd^.interpret := "";   (* Initialisiere Interpreten *)
  newcd^.title := "";       (* Initialisiere Titel *)
  newcd^.tracklist := NIL;  (* Initialisiere Trackliste *)
  RETURN newcd;
END CreateCD;

BEGIN
END CD.
```

```
MODULE Track;

(* Modulrumpf des abstrakten Datentyps Track.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 02.01.01
*)


(* Ein Track besteht aus einem Namen und der Länge in Sekunden *)
TYPE REVEAL T = BRANDED REF RECORD
                  name: TEXT;
                  length: CARDINAL;
                END;


PROCEDURE GetName(track: T): TEXT =
(* Liefert den Namen des Tracks *)
BEGIN
  RETURN (track^.name);
END GetName;


PROCEDURE SetName(VAR track: T; n: TEXT) =
(* Setzt den Namen des Tracks auf den übergebenen String *)
BEGIN
  track^.name := n;
END SetName;


PROCEDURE GetLength(track: T): CARDINAL =
(* Liefert die Länge des Tracks in Sekunden *)
BEGIN
  RETURN (track^.length);
END GetLength;


PROCEDURE SetLength(VAR track: T; l: CARDINAL) =
(* Setzt die Länge des übergebenen Tracks in Sekunden *)
BEGIN
  track^.length := l;
END SetLength;
```

```
PROCEDURE CreateTrack(): T =
(* Erstellt einen neuen Track und initialisiert diesen *)
VAR newtrack: T;
BEGIN
  newtrack := NEW(T);
  newtrack^.name := "";   (* Initialisiere den Namen *)
  newtrack^.length := 0;  (* Initialisiere die Länge *)
  RETURN newtrack;
END CreateTrack;

BEGIN
END Track.
```

```
MODULE Diskographie;

(* Dieses Module implementiert die Diskographie-Verwaltung, wie
in der Schnittstelle
   Diskographie definiert. Die Verwaltung erfolgt mit Hilfe eines
Feldes begrenzter Kapazitaet.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 19.12.00
*)

IMPORT SIO;
IMPORT CD;   (* Importiere den ADT CD *)

CONST Max = 5;

TYPE DiskographieIndex = [0 .. Max];
     Diskographieverwaltung = ARRAY [1 .. Max] OF CD.T;

VAR index : DiskographieIndex;
    verwaltung: Diskographieverwaltung;


PROCEDURE Init()=
(* Initialisiert die Diskographie. Diese enthält dann keine CDs
*)
BEGIN
  index := 0;
END Init;


PROCEDURE AddCD(cd: CD.T)=
(* Fügt die übergebene CD der Diskographie hinzu *)
BEGIN
  (* CD wird im nächsten Speicherplatz abgelegt *)
  IF NOT IsFull() THEN
    index := index + 1;
    verwaltung[index] := cd;
```

```
  ELSE
    SIO.PutLine("Diskographie ist voll!!!");
  END;
END AddCD;


PROCEDURE RemoveCD(pos: CARDINAL)=
(* Löscht die CD an der übergebenen Indexposition *)
BEGIN
  IF IsEmpty() THEN
    SIO.PutLine("Keine CD zum loeschen vorhanden!!!");
  ELSE
    IF (pos > 0) AND (pos <= index) THEN
      FOR i:=pos TO index - 1 DO
        verwaltung[i] := verwaltung[i+1];
      END;
      (* Indexposition wird um 1 zurückgesetzt *)
      index := index - 1;
    END;
  END;
END RemoveCD;


PROCEDURE IsFull(): BOOLEAN =
(* Prüft, ob die Diskographie voll ist *)
BEGIN
  RETURN (index = Max);
END IsFull;


PROCEDURE IsEmpty(): BOOLEAN =
(* Prüft, ob die Diskographie leer ist *)
BEGIN
  RETURN (index = 0);
END IsEmpty;


PROCEDURE Print()=
(* Listet den Inhalt der Diskographie auf dem Bildschirm *)
BEGIN
  SIO.PutLine("------------------- Diskographieverwaltung ------
----------------------------------------");
  SIO.Nl();
  FOR i:=1 TO index DO
    SIO.PutInt(i); SIO.PutText(".   ");
    SIO.PutText("Titel: ");
SIO.PutText(CD.GetTitle(verwaltung[i]));
    SIO.PutText("   Interpret: ");
SIO.PutText(CD.GetInterpret(verwaltung[i]));
    SIO.PutText("   Anzahl Tracks: ");
SIO.PutInt(CD.GetTrackCount(verwaltung[i]));
    SIO.PutText("   Gesamtspieldauer: ");
SIO.PutInt(CD.GetTotalTime(verwaltung[i]));
    SIO.Nl();
  END;
  SIO.PutLine("------------------------------------------------
----------------------------------------");
END Print;


PROCEDURE PrintWithTracklist()=
(* Listet den Inhalt der Diskographie mit den Tracks jeder CD auf
dem Bildschirm *)
BEGIN
  SIO.PutLine("------------------- Diskographieverwaltung ------
----------------------------------------");
  SIO.Nl();
  FOR i:=1 TO index DO
    SIO.PutInt(i); SIO.PutText(".   ");
    SIO.PutText("Titel: ");
SIO.PutText(CD.GetTitle(verwaltung[i]));
    SIO.PutText("   Interpret: ");
SIO.PutText(CD.GetInterpret(verwaltung[i]));
    SIO.PutText("   Anzahl Tracks: ");
SIO.PutInt(CD.GetTrackCount(verwaltung[i]));
    SIO.PutText("   Gesamtspieldauer: ");
SIO.PutInt(CD.GetTotalTime(verwaltung[i]));
    SIO.Nl();
    SIO.PutLine("Tracks:");
    CD.PrintTracklist(verwaltung[i]);
    SIO.Nl();
  END;
  SIO.PutLine("------------------------------------------------
----------------------------------------");
END PrintWithTracklist;

PROCEDURE GetCD(i: CARDINAL): CD.T =
(* Liefert die CD der Diskographie mit dem übergebenen Index *)
BEGIN
  IF (i > 0) AND (i <= index) THEN
    RETURN (verwaltung[i]);
  ELSE
    SIO.PutLine("Keine gueltige CD angegeben");
    RETURN NIL;
  END;
END GetCD;

BEGIN
END Diskographie.
```

```
MODULE CDVerwaltung EXPORTS Main;

(* Hauptprogramm der CD-Verwaltung. Das Programm stellt ein Menü
bereit, welches es ermöglicht CDs in die Diskographie einzufügen
```

```
und zu löschen. Ebenso können Musikstücke zu den verschiedenen
enthaltenen CDs eingefügt und auch wieder gelöscht werden.
   Autor           : Thomas von der Maßen, RWTH Aachen
   Umgebung        : PM-3 Windows 2000
   Erstellt        : 19.12.00  Letzte Aenderung: 02.01.01
*)

IMPORT SIO;
IMPORT Diskographie;  (* Importiere ADO Diskographie *)
IMPORT CD;            (* Importiere ADT CD *)
IMPORT Track;         (* Importiere ADT Track *)

VAR dummy : TEXT;
    auswahl : CHAR;
    cd : CD.T;
    track: Track.T;
    nummer: CARDINAL;


PROCEDURE PrintMenu()=
(* Hilfsprozedur, die das Befehlsmenue ausgibt *)
BEGIN
  SIO.Nl();
  SIO.PutLine("F : Fuege neue CD ein");
  SIO.PutLine("L : Loesche CD");
  SIO.PutLine("Z : Zeige Diskographie an");
  SIO.PutLine("T : Fuege Track in CD ein");
  SIO.PutLine("X : Loesche Track von CD");
  SIO.PutLine("E : Exit");
  SIO.Nl();
END PrintMenu;


BEGIN
  Diskographie.Init();   (* Initialisiere die Diskographie *)

  REPEAT
    PrintMenu();

    (* Frage Benutzer nach seiner Auswahl *)
    SIO.PutText("Auswahl: ");
    auswahl := SIO.GetChar();
    dummy := SIO.GetLine(); (* Lies RETURN aus dem Eingabepuffer!
*)
    SIO.Nl();

    CASE auswahl OF
    (* Erstellt eine neue CD *)
    |  'F', 'f' => cd := CD.CreateCD();
                   SIO.PutText("Bitte geben Sie den CD-Titel ein:
");
                   CD.SetTitle(cd, SIO.GetLine());
                   SIO.PutText("Bitte geben Sie den Interpreten
ein: ");
                   CD.SetInterpret(cd, SIO.GetLine());
                   Diskographie.AddCD(cd);

    (* Löschen der letzten CD *)
    |  'L', 'l' => Diskographie.Print();
                   SIO.PutText("Geben Sie die CD-Nummer ein: ");
                   nummer := SIO.GetInt();
                   dummy := SIO.GetLine();
                   Diskographie.RemoveCD(nummer)

    (* Ausgeben der aktuellen Diskographie *)
    |  'Z', 'z' => Diskographie.PrintWithTracklist();

    (* Einen Track einer CD hinzufügen *)
    |  'T', 't' => Diskographie.Print();
                   SIO.PutText("Geben Sie die CD-Nummer ein: ");
                   nummer := SIO.GetInt();
                   dummy := SIO.GetLine();
                   cd := Diskographie.GetCD(nummer);
                   IF (cd # NIL) THEN
                     REPEAT
                       track := Track.CreateTrack();
                       SIO.PutText("Geben sie den Namen des
Tracks ein: ");
                       Track.SetName(track, SIO.GetLine());
                       SIO.PutText("Geben Sie die Laenge des
Tracks in Sekunden ein: ");
                       Track.SetLength(track, SIO.GetInt());
                       dummy := SIO.GetLine();
                       CD.AddTrack(cd, track);
                       SIO.PutText("Moechten Sie einen weiteren
Track eingeben (j/n)?");
                       auswahl := SIO.GetChar();
                       dummy := SIO.GetLine();
                     UNTIL (auswahl = 'n') OR (auswahl = 'N');
                   END;

    (* Einen Track von einer CD löschen *)
    |  'X', 'x' => Diskographie.Print();
                   SIO.PutText("Geben Sie die CD-Nummer ein: ");
                   nummer := SIO.GetInt();
                   dummy := SIO.GetLine();
                   cd := Diskographie.GetCD(nummer);
                   IF (cd # NIL) THEN
                     REPEAT
                       CD.RemoveTrack(cd);
                       SIO.PutText("Moechten Sie einen weiteren
Track loeschen (j/n)?");
                       auswahl := SIO.GetChar();
                       dummy := SIO.GetLine();
                     UNTIL (auswahl = 'n') OR (auswahl ='N');
```

```
                    END;

    (* Das Programm beenden *)
    |  'E', 'e' => (* Nichts machen, gleich ist ohnehin alles zu
Ende! *)
    ELSE
      SIO.PutLine("Ungueltiger Befehl!");
    END (* CASE *)
  UNTIL (auswahl = 'e') OR (auswahl = 'E');
END CDVerwaltung.
```

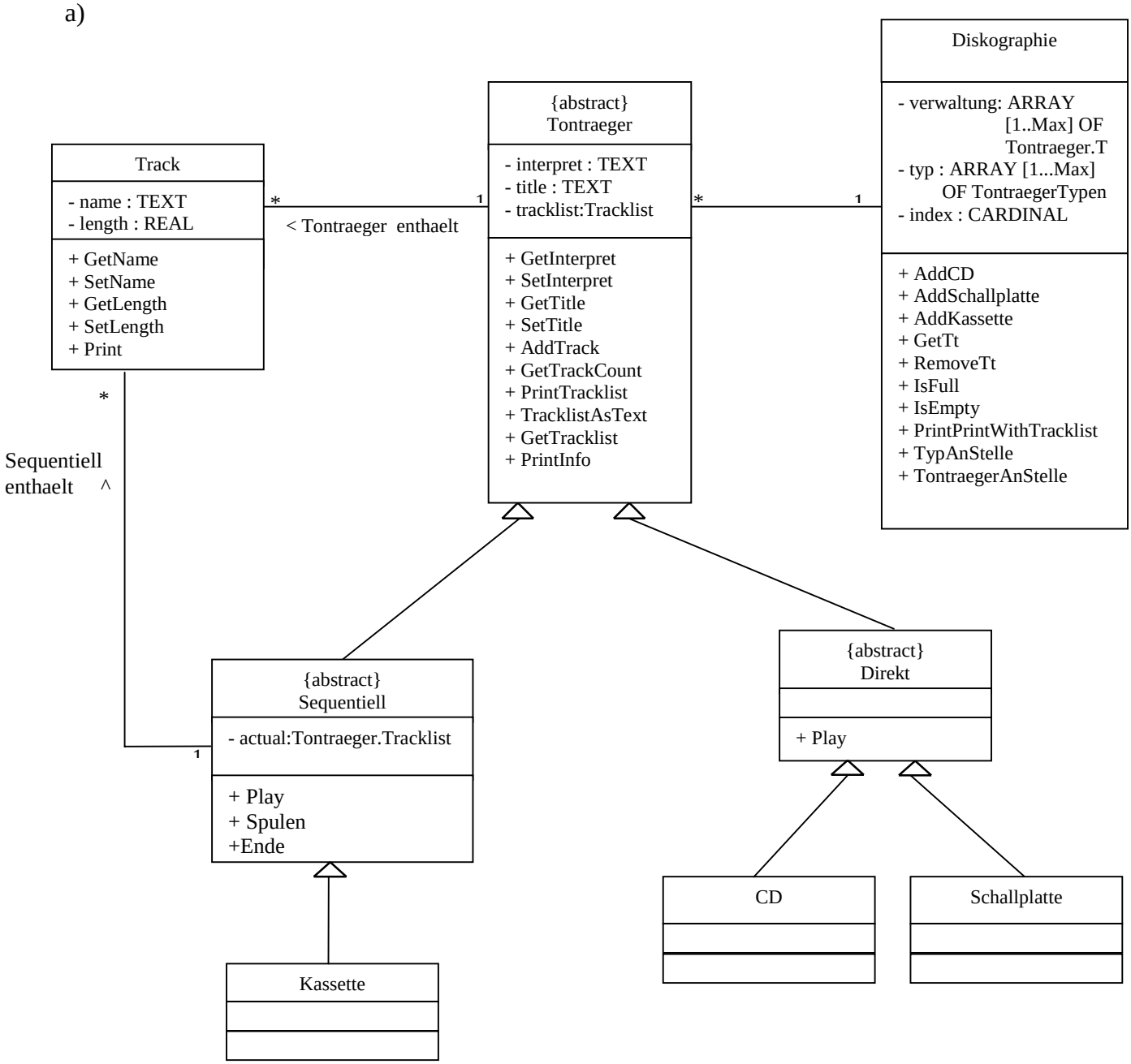
Programmierung **WS 2000 / 2001** **Prof. H. Lichter**
A. Nowack

Übung 11

Musterlösung

Aufgabe 11.1:

a)



Anmerkung:
Dies ist nur eine Möglichkeit, die Aufgabenstellung zu realisieren. Der Typ Trackliste ist definiert im Interface Tontraeger, eine Kapselung als Klasse bzw. eine Kapselung des Typs der Listenelemente sowie die Einfuehrung eines Containers wäre in objekt-orientierten

Sprachen notwendig. In Modula-3 (einer hybriden Sprache) ist dies nicht der Fall (wäre aber ein anderer möglicher Ansatz).

b)

```
INTERFACE Track;

TYPE T <: Public;
   Public = ROOT OBJECT
          METHODS
               GetName(): TEXT ;
               SetName(name: TEXT);
               GetLength(): CARDINAL ;
               SetLength(length: CARDINAL);
               Print();
          END;

END Track.
```

```
MODULE Track;

IMPORT SIO;

REVEAL
   T = Public BRANDED OBJECT
               name: TEXT;
               length: CARDINAL;
             OVERRIDES
               GetName := GetNameTrack;
               SetName := SetNameTrack;
               GetLength := GetLengthTrack;
               SetLength := SetLengthTrack;
               Print := PrintTrack;

    END;


PROCEDURE GetNameTrack(self: T): TEXT =
BEGIN
  RETURN self.name;
END GetNameTrack;

PROCEDURE SetNameTrack(self: T; n: TEXT) =
BEGIN
  self.name := n;
END SetNameTrack;
```

Entsprechend:
PROCEDURE GetLengthTrack(self: T): CARDINAL
PROCEDURE SetLengthTrack(self: T; l: CARDINAL)
PROCEDURE PrintTrack(self : T)

```
BEGIN
END Track.
```

```
INTERFACE Tontraeger;

IMPORT Track;

TYPE Tracklist = REF RECORD
                   track: Track.T;
                   next: Tracklist;
                 END;

TYPE T <: Public;
   Public = ROOT OBJECT
              METHODS
                    GetInterpret(): TEXT;
                    SetInterpret(interpret: TEXT);
                    GetTitle(): TEXT;
                    SetTitle(title: TEXT);
                    AddTrack(track: Track.T);
                    GetTrackCount(): CARDINAL;
                    PrintTracklist();
                    TracklistAsText() : TEXT;
                    GetTracklist() : Tracklist;
                    PrintInfo(): TEXT;
               END;

END Tontraeger.
```

```
MODULE Tontraeger;

IMPORT Track;
IMPORT SIO;
IMPORT Text;
IMPORT Fmt;

REVEAL
      T = Public BRANDED OBJECT
                  interpret: TEXT;
                  title: TEXT;
                  tracklist: Tracklist := NIL;
            OVERRIDES
                    GetInterpret := GetInterpretTt;
                    SetInterpret := SetInterpretTt;
                    GetTitle := GetTitleTt;
                    SetTitle := SetTitleTt;
                    AddTrack := AddTrackTt;
                    GetTrackCount := GetTrackCountTt;
                    PrintTracklist := PrintTracklistTt;
                    GetTracklist := GetTracklistTt;
```

```
                    PrintInfo := PrintTt;
                    TracklistAsText := TracklistAsTextTt;
          END;

PROCEDURE GetInterpretTt(self: T): TEXT =
BEGIN
  RETURN (self.interpret);
END GetInterpretTt;

PROCEDURE SetInterpretTt(self: T; interpret: TEXT) =
BEGIN
  self.interpret := interpret;
END SetInterpretTt;
```

Entsprechend:
PROCEDURE GetTitleTt(self: T): TEXT
PROCEDURE SetTitleTt(self: T; title: TEXT)

```
PROCEDURE AddTrackTt(self: T; track: Track.T) =
(* Fügt einen Track dem Tontraeger hinzu. Der neue Track wird
   an das Ende der Trackliste angehängt. *)
VAR newtrack, actual, prev: Tracklist;
VAR add : BOOLEAN := TRUE;
BEGIN
  newtrack := NEW(Tracklist);
  newtrack^.track := track;
  newtrack^.next := NIL;

  (* Falls Trackliste leer *)
  IF (self.tracklist = NIL) THEN
    self.tracklist := newtrack;
  ELSE
    (* Suche Ende der Trackliste *)
    actual := self.tracklist;
    prev := self.tracklist;
    WHILE (actual # NIL) DO
      (* Falls der einzufuegende Track bereits an der
         betrachteten Stelle steht *)
      IF Text.Equal(track.GetName() ,
                    (actual^.track).GetName()) AND
         track.GetLength() = (actual^.track).GetLength()
      THEN(* wird der Tontraeger nicht veraendert. *)
        actual := NIL;
        add := FALSE;
        SIO.PutLine("Der angegebene Track ist bereits
                     enthalten.");
      ELSE
        (* sonst Aenderung gemaess obiger Beschreibung *)
        prev := actual;
        actual := actual^.next;
      END;
    END;
```

```
    IF add THEN prev^.next := newtrack; END;
  END;
END AddTrackTt;
```

Entsprechend:
PROCEDURE GetTrackCountTt(self: T): CARDINAL
PROCEDURE PrintTracklistTt(self: T)
PROCEDURE GetTracklistTt(self : T) : Tracklist
PROCEDURE TracklistAsTextTt(self : T) : TEXT
(* Liefert die Trackliste des Tontraegers mit Namen und Länge der Tracks als Text. Dies wird realisiert durch einen rekursiven Durchlauf durch die Liste. *)

```
PROCEDURE PrintTt(self : T) : TEXT =
(* Liefert einen Text, der alle Informationen - inklusive
   Trackliste - ueber den Tontraeger enthaelt. *)
BEGIN
   RETURN "Interpret: " & self.interpret & "   Titel: " &
          self. title  & "\n" & self.TracklistAsText() ;
END PrintTt;

BEGIN
END Tontraeger.
```

```
INTERFACE Sequentiell;

IMPORT Tontraeger;
IMPORT Track;

TYPE T <: Public;
   Public = Tontraeger.T OBJECT
                  METHODS
                    Play() : Track.T;
                    (* Abspielen des sequentiellen
                       Tontraegers an der aktuellen Stelle *)
                    Spulen();
                    (* Spulen des Tontraegers *)
                    Ende(): BOOLEAN;
                    (* Ueberprueft, ob der Tontraeger an
                       seinem Ende angelangt ist. *)
                  END;

END Sequentiell.
```

```
MODULE Sequentiell;

IMPORT Track;
IMPORT Tontraeger;
IMPORT SIO;
IMPORT Text;
IMPORT Fmt;
```

```
REVEAL
   T = Public BRANDED OBJECT
                      actual: Tontraeger.Tracklist := NIL;
                   OVERRIDES
                    Play := PlayerSeq;
                    (* Abspielen des sequentiellen Tontraegers
                       an der aktuellen Stelle *)
                    Spulen := SpulenSeq;
                    PrintInfo := PrintSeq;
                    (* Dies ist eine Redefinition der
                       entsprechenden Methode des Typs
                       Tontraeger. *)
                    AddTrack := AddTrackSeq;
                    (* Fügt einen Track des übergebenen
                       Tontraegers hinzu. Dies ist eine
                       Redefinition der entsprechenden Methode
                       des Typen Tontraeger. *)
                    Ende := EndeSeq;
            END;


PROCEDURE SpulenSeq(self: T) =
VAR act : Tontraeger.Tracklist;
BEGIN
(* Die Trackliste wird von Anfang an so lange durchlaufen,
   bis der Vorgraenger des aktuellen Tracks gefunden
   ist. Der aktuelle Trakc wird durch seinen Vorgaenger
   ersetzt. *)
   act := self.GetTracklist();
  (* Wenn der aktuelle Track nicht der erste Track ist und
    somit der Track keinen Vorgraenger hat, der Tontraeger
    also nicht bereits am Anfang steht ... *)
  IF NOT self.Ende() THEN
    IF NOT ( Text.Equal( ((self.actual)^.track).GetName(),
                  ((self.GetTracklist())^.track).GetName())
           AND ( ((self.actual)^.track).GetLength () =
               ((self.GetTracklist())^.track).GetLength()))
    THEN
    (* ...dann suche den Vorgaenger des aktuellen Tracks ...*)
      WHILE  NOT (Text.Equal( ((act^.next)^.track).GetName(),
                              ((self.actual)^.track).GetName())
                   AND ( ((act^.next)^.track).GetLength () =
                       ((self.actual)^.track).GetLength())) DO
           act := act^.next;
        END;
        self.actual :=act;
        (* ... sonst ist Spulen nicht moeglich. *)

        ELSE SIO.PutLine("Spulen nicht moeglich! Der
                          Tontraeger ist bereits am Anfang.");
    END;
```

```
  ELSIF self.GetTracklist() = NIL
  THEN SIO.PutLine("Die Trackliste ist leer!");
  ELSE
    WHILE  NOT act.next = NIL DO
          act := act^.next;
        END;
        self.actual :=act;
  END;
END SpulenSeq;

PROCEDURE PlayerSeq(self : T) : Track.T =
VAR play: Tontraeger.Tracklist;
BEGIN
  (* Liefere den aktuellen Track zurueck ... *)
  play := self.actual;
  (* ... und ersetze den aktuellen Track durch seinen
     Vorgaenger ... *)
  self.actual := (self.actual)^.next;
  RETURN play.track;
END PlayerSeq;

PROCEDURE EndeSeq(self:T) : BOOLEAN =
BEGIN
  RETURN self.actual = NIL;
END EndeSeq;

PROCEDURE PrintSeq(self : T) : TEXT =
BEGIN
  (* Liefert zusaetzlich zu den Informationen, die die Methode
     des Supertypen Tontraeger liefert, die Zugriffsart. *)
  IF NOT self.actual = NIL THEN
    RETURN Tontraeger.T.PrintInfo(self) &   " Zugriff :
           Sequentiell  " & "\n" & "       " & " aktueller
           Track: " & (((self.actual)^.track).GetName())
           & "   Laenge: " & Fmt.Int
                     ((((self.actual)^.track).GetLength ()))
           & " Sekunden" &"\n" & "    ";
  ELSE
    RETURN Tontraeger.T.PrintInfo(self) &   " Zugriff :
           Sequentiell  " & "\n" & "       " & "Der Tontaeger
           befindet sich am Ende." &"\n" & "    ";
  END;
END PrintSeq;
```

Entsprechend: PROCEDURE AddTrackSeq(self: T; track: Track.T)
(* Fügt einen Track des übergebenen Tontraegers hinzu. Dies realisiert die Redefinition der entsprechenden Methode des Supertypen Tontraeger. Ausser der Aktualisierung der Trackliste ist hier auch die Aktualisierung der Liste notwendig, die den aktuellen Track angibt. Deren Aktualisierung wird genauso realisiert wie die der Trackliste. *)

```
BEGIN
```

```
END Sequentiell.
```

```
INTERFACE Direkt;

IMPORT Tontraeger;
IMPORT Track;

TYPE T <: Public;
   Public = Tontraeger.T  OBJECT
               METHODS
                 Play(pos : CARDINAL) : Track.T;
                 (* Abspielen des Tontraegers mit direktem
                    Zugriff. Der Track an der n-ten Position
                    der Trackliste wird zurueckgelierfert. *)
               END;

END Direkt.
```

```
MODULE Direkt;

IMPORT Tontraeger;
IMPORT Track;

REVEAL
   T = Public BRANDED OBJECT
                OVERRIDES
                  Play := PlayerDir;
                  (* Abspielen des Tontraegers mit direktem
                     Zugriff. Der Track an der n-ten Position
                     Der Trackliste wird zurueckgelierfert. *)
                  PrintInfo  := PrintDir;
                  (* Dies ist eine Redefinition der
                     entsprechenden Methode des Typs
                     Tontraeger. *)
        END;

PROCEDURE PlayerDir(self : T; n: CARDINAL) : Track.T =
VAR liste: Tontraeger.Tracklist;
BEGIN
   liste := Tontraeger.T.GetTracklist(self);
   (* Suche den Track an der n-ten Stelle ... *)
   FOR i := 1 TO (n-1)  DO
       liste := liste^.next;
   END;
   (* ... und liefere diesen zurueck. *)
   RETURN liste.track;
END PlayerDir;

PROCEDURE PrintDir(self : T) : TEXT =
BEGIN
```

```
(* Liefert zusaetzlich zu den Informationen, die die Methode
   des Supertypen Tontraeger liefert, die Zugriffsart. *)

  RETURN Tontraeger.T.PrintInfo(self) & " Zugriff : Direkt"
         &"\n" & "    ";
END PrintDir;

BEGIN
END Direkt.
```

```
INTERFACE Kassette;

IMPORT Sequentiell;

TYPE T <: Public;
   Public = Sequentiell.T OBJECT
             END;
END Kassette.
```

```
MODULE Kassette;

IMPORT Sequentiell;

REVEAL
   T = Public BRANDED OBJECT
     OVERRIDES
         PrintInfo := PrintKassette;
     END;

PROCEDURE PrintKassette(self : T) : TEXT =
BEGIN
  RETURN Sequentiell.T.PrintInfo(self) & " Typ : Kassette";
END PrintKassette;

BEGIN
END Kassette.
```

```
INTERFACE CD;

IMPORT Direkt;

TYPE T <: Public;
   Public = Direkt.T  OBJECT
            END;

END CD.
```

```
MODULE CD;

IMPORT Direkt;

REVEAL
   T = Public BRANDED OBJECT
          OVERRIDES
            PrintInfo := PrintCD;
            (* Ausgabe aller Informationen ueber den
               Tontraeger als Text. Dies ist eine Redefinition
               der entsprechenden Methode des Typen Direkt. *)
          END;

PROCEDURE PrintCD(self :T) : TEXT =
BEGIN
  RETURN Direkt.T.PrintInfo(self) & " Typ : CD";
END PrintCD;


BEGIN
END CD.
```

```
INTERFACE Schallplatte;

IMPORT Direkt;
TYPE T <: Public;
   Public = Direkt.T OBJECT
  END;
END Schallplatte.
```

```
MODULE Schallplatte;

IMPORT Direkt;

REVEAL
   T = Public BRANDED OBJECT
          OVERRIDES
            PrintInfo := PrintSchallplatte;
            (* Dies ist eine Redefinition der entsprechenden
               Methode des Typen Direkt. *)
          END;

PROCEDURE PrintSchallplatte(self : T) : TEXT =
BEGIN
  RETURN Direkt.T.PrintInfo(self) & " Typ : Schallplatte";
END PrintSchallplatte;

BEGIN
END Schallplatte.
```

```
INTERFACE Diskographie;

IMPORT Tontraeger;
IMPORT CD;
IMPORT Schallplatte;
IMPORT Kassette;

TYPE TontraegerTypen = {CDTyp, SchallplatteTyp, KassetteTyp};

TYPE T <: Public;
   Public = ROOT OBJECT
              METHODS
                AddCD(cd: CD.T);
                AddSchallplatte(schp: Schallplatte.T);
                AddKassette(kassette: Kassette.T);
                GetTt(index: CARDINAL): Tontraeger.T;
                RemoveTt(pos: CARDINAL;);
                IsFull(): BOOLEAN;
                IsEmpty(): BOOLEAN;
                Print();
                PrintWithTracklist();
                TypAnStelle(i : CARDINAL) : TontraegerTypen;
                TontraegerAnStelle(i : CARDINAL):Tontraeger.T;
              END;

END Diskographie.
```

```
MODULE Diskographie;

IMPORT SIO;
IMPORT CD;   (* Importiere den ADT CD *)
IMPORT Tontraeger;
IMPORT Schallplatte;
IMPORT Kassette;

CONST Max = 5;

REVEAL
      T = Public BRANDED OBJECT
                  verwaltung:  ARRAY [1 .. Max] OF
                               Tontraeger.T;
                  typ : ARRAY [1..Max] OF TontraegerTypen;
                  index : CARDINAL := 0;
            OVERRIDES
                AddCD := AddCDDisk;
                AddSchallplatte := AddSchallplatteDisk;
                AddKassette := AddKassetteDisk;
                GetTt := GetTtDisk;
                RemoveTt:= RemoveTtDisk;
                IsFull := IsFullDisk;
```

```
                IsEmpty := IsEmptyDisk;
                Print := PrintDisk;
                PrintWithTracklist := PrintWithTracklistDisk;
                TypAnStelle := TypAnStelleDisk;
                TontraegerAnStelle := TontraegerAnStelleDisk;

            END;

PROCEDURE AddCDDisk(self: T;  cd: CD.T)=
BEGIN
  (* CD wird im nächsten Speicherplatz abgelegt *)
  IF NOT self.IsFull() THEN
    self.index := self.index + 1;
    self.verwaltung[self.index] := cd;
    self.typ[self.index] := TontraegerTypen.CDTyp;
  ELSE
    SIO.PutLine("Diskographie ist voll!!!");
  END;
END AddCDDisk;
```

Entsprechend:
PROCEDURE AddSchallplatteDisk(self: T; sch: Schallplatte.T)
PROCEDURE AddKassetteDisk(self: T; kass: Kassette.T)
PROCEDURE RemoveTtDisk(self:T; pos: CARDINAL)
PROCEDURE IsFullDisk(self: T): BOOLEAN
PROCEDURE IsEmptyDisk(self:T): BOOLEAN
PROCEDURE PrintDisk(self : T)=

```
PROCEDURE PrintWithTracklistDisk(self:T)=
BEGIN
  SIO.PutLine(" *** Diskographieverwaltung *** ");
  SIO.Nl();
  FOR i:=1 TO self.index DO
    SIO.PutInt(i); SIO.PutText(".   ");
    IF self.typ[i] = TontraegerTypen.CDTyp
    THEN SIO.PutLine(CD.T.PrintInfo(self.verwaltung[i])) END;
    IF self.typ[i] = TontraegerTypen.SchallplatteTyp
    THEN
     SIO.PutLine(Schallplatte.T.PrintInfo(self.verwaltung[i]))
    END;
    IF self.typ[i] = TontraegerTypen.KassetteTyp
    THEN SIO.PutLine(Kassette.T.PrintInfo(self.verwaltung[i]))
END;
  END;
END PrintWithTracklistDisk;
```

Entsprechend:
PROCEDURE GetTtDisk(self: T; i: CARDINAL): Tontraeger.T =
PROCEDURE TypAnStelleDisk(self : T; i : CARDINAL) : TontraegerTypen =
PROCEDURE TontraegerAnStelleDisk(self: T; i: CARDINAL): Tontraeger.T =

```
BEGIN
```

```
END Diskographie.
```

```
MODULE Verwaltung EXPORTS Main;

IMPORT SIO;
IMPORT Diskographie;  (* Importiere ADO Diskographie *)
IMPORT Tontraeger;
IMPORT CD;            (* Importiere ADT CD *)
IMPORT Schallplatte;
IMPORT Kassette;
IMPORT Track;         (* Importiere ADT Track *)

VAR dummy : TEXT;
    auswahl : CHAR;
    cd : CD.T;
    sch : Schallplatte.T;
    kass : Kassette.T;
    tt : Tontraeger.T;
    track: Track.T;
    nummer: CARDINAL;
    disk : Diskographie.T;
    i,j : CARDINAL;
```

Die Implementierung folgender Hilfsprozedur zur Ausgabe des Menüs ist offensichtlich:
PROCEDURE PrintMenu()

```
BEGIN
  disk := NEW(Diskographie.T);

  REPEAT
    PrintMenu();

    (* Frage Benutzer nach seiner Auswahl *)
    SIO.PutText("Auswahl: ");
    auswahl := SIO.GetChar();
    dummy := SIO.GetLine();
    (* Lies RETURN aus dem Eingabepuffer! *)
    SIO.Nl();

    CASE auswahl OF
    (* Erstellt eine neue CD *)
    |  'c', 'C' => cd :=NEW( CD.T);
                   SIO.PutText("Bitte geben Sie den CD-Titel
                                ein: ");
                   cd.SetTitle(SIO.GetLine());
                   SIO.PutText("Bitte geben Sie den
                                Interpreten ein: ");
                   cd.SetInterpret(SIO.GetLine());
                   disk.AddCD(cd);

    |  's', 'S' => ... Erstellung einer Schallplatte ... wie für CDs
```

```
  |  'k', 'K' => ... Erstellung einer Kassette ... wie für CDs


| 'L', 'l' =>  SIO.PutLine("Bitte geben Sie die Nummer des
                          Tontraegers ein: ");
             i := SIO.GetInt();
             disk.RemoveTt(i);


  |  'Z', 'z' => disk.PrintWithTracklist();

  |  'T', 't' => Diskographie.T.Print(disk);
             SIO.PutText("Geben Sie die Tontrager-Nummer
                          ein: ");
             nummer := SIO.GetInt();
             dummy := SIO.GetLine();
             tt := disk.GetTt(nummer);
             IF (tt # NIL) THEN
               REPEAT
                 SIO.PutText("Möchten Sie einen Track
                              eingeben (j/n)?");
                 auswahl := SIO.GetChar();
                 dummy := SIO.GetLine();
                 IF (auswahl = 'j') OR (auswahl = 'J')
                 THEN
                   track := NEW(Track.T);
                   SIO.PutText("Geben sie den Namen des
                                Tracks ein: ");
                   track.SetName(SIO.GetLine());
                   SIO.PutText("Geben Sie die Laenge des
                            Tracks in Sekunden ein: ");
                   track.SetLength(SIO.GetInt());
                   dummy := SIO.GetLine();
                   tt.AddTrack(track);
                 END;
               UNTIL (auswahl = 'n') OR (auswahl = 'N');
             END;


  | 'P', 'p' => Diskographie.T.Print(disk);
             SIO.PutLine("Bitte die Nummer des
                          Tontraegers eingeben: ");
             i := SIO.GetInt();
             IF disk.TypAnStelle(i) =
                      Diskographie.TontraegerTypen.CDTyp
             THEN
               cd := disk.TontraegerAnStelle(i);
               SIO.PutLine("Bitte die Tracknummer
                            eingeben:");
               j := SIO.GetInt();
```

```
                    (cd.Play(j)).Print();
                  ELSIF disk.TypAnStelle(i) =
                           Diskographie.
                             TontraegerTypen.SchallplatteTyp
                  THEN
                    Abspielen einer Schallplatte ... wie für CDs
                  THEN
                    kass := disk.TontraegerAnStelle(i);
                    IF kass.Ende()
                    THEN SIO.PutLine("Die Kassette ist am
                                      Ende. Abspielen nicht
                                      moeglich!");
                    ELSE
                       (kass.Play()).Print();
                     END;
                  END;
    | 'B', 'b' => Diskographie.T.Print(disk);
                     SIO.PutLine("Bitte geben Sie die Nummer
                                  des Tontraegers ein");
                     i := SIO.GetInt();
                     IF disk.TypAnStelle(i) =
                         Diskographie.TontraegerTypen.
                           KassetteTyp
                     THEN
                        kass := disk.TontraegerAnStelle(i);
                        kass.Spulen();
                     ELSE
                        SIO.PutLine("Der Tontraeger hat
                                     keinen sequentiellem
                                     Zugriff!");
                     END;

    (* Das Programm beenden *)
    |  'E', 'e' => (* Nichts machen, gleich ist ohnehin alles
                     zu Ende! *)
    ELSE
      SIO.PutLine("Ungueltiger Befehl!");
    END (* CASE *)
  UNTIL (auswahl = 'e') OR (auswahl = 'E');
END Verwaltung.
```

Testlauf:

C : Fuege neue CD ein
S : Fuege neue Schallplatte ein
K : Fuege neue Kassette ein
L : Loesche Tontraeger
Z : Zeige Diskographie an
T : Fuege Track in Tontraeger ein
X : Loesche Track von Tontraeger
P : Track von einem Tontraeger spielen
B : Spule Kassette zurueck

E : Exit

Auswahl: c, dann Auswahl: s, dann Auswahl: k jeweils mit Eingabe, so daß sich die untenstehende Diskographie ergibt

... Menü ...
Auswahl: t
*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 0
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 0
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks: 0

Geben Sie die Tontrager-Nummer ein: 1
Moechten Sie einen Track eingeben (j/n)?j
Geben sie den Namen des Tracks ein: cdtrack
Geben Sie die Laenge des Tracks in Sekunden ein: 1
Moechten Sie einen Track eingeben (j/n)?n

... Menü ...

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 0
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks: 0

Geben Sie die Tontrager-Nummer ein: 2
Moechten Sie einen Track eingeben (j/n)?j
Geben sie den Namen des Tracks ein: schallplattentrack
Geben Sie die Laenge des Tracks in Sekunden ein: 2
Moechten Sie einen Track eingeben (j/n)?n

... Menü ...
Auswahl: t

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 1
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks: 0

Geben Sie die Tontraeger-Nummer ein: 3
Moechten Sie einen Track eingeben (j/n)?j
Geben sie den Namen des Tracks ein: kassettentrack1
Geben Sie die Laenge des Tracks in Sekunden ein: 4
Moechten Sie einen Track eingeben (j/n)?j
Geben sie den Namen des Tracks ein: kassettentrack2
Geben Sie die Laenge des Tracks in Sekunden ein: 5
Moechten Sie einen Track eingeben (j/n)?n

... Menü ...
Auswahl: z

*** Diskographieverwaltung ***

1. Interpret: CDinterpret Titel: CD
   1. CDtrack Dauer: 1 Sekunden

   Zugriff : Direkt
   Typ : CD
2. Interpret: Schallplatteninterpret Titel: Schallplatte
   1. Schallplattentrack Dauer: 2 Sekunden

   Zugriff : Direkt
   Typ : Schallplatte
3. Interpret: Kassetteninterpret Titel: Kassette
   1. Kassettentrack1 Dauer: 4 Sekunden
   2. Kassettentrack2 Dauer: 5 Sekunden

   Zugriff : Sequentiell
   aktueller Track: Kassettentrack1 Laenge: 4 Sekunden
   Typ : Kassette

... Menü ...
Auswahl: P

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 1
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks: 2

Bitte die Nummer des Tontraegers eingeben:
1
Bitte die Tracknummer eingeben:
1
Titel: CDtrack Laenge: 1 Sekunden

... Menü ...
Auswahl: p

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 1
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks: 2

Bitte die Nummer des Tontraegers eingeben:
3
Titel: Kassettentrack1 Laenge: 4 Sekunden

... Menü ...
Auswahl: p

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 1
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks: 2

Bitte die Nummer des Tontraegers eingeben:
3
Titel: Kassettentrack2 Laenge: 5 Sekunden

... Menü ...
Auswahl: b

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 1
3. Titel: Kassetteninterpret Interpret: Kassetteninterpret Anzahl Tracks: 2

Bitte geben Sie die Nummer des Tontraegers ein
3

... Menü ...
Auswahl: p

*** Diskographieverwaltung ***

1. Titel: CD Interpret: CDinterpret Anzahl Tracks: 1
2. Titel: Schallplatte Interpret: Schallplatteninterpret Anzahl Tracks: 1
3. Titel: Kassette Interpret: Kassetteninterpret Anzahl Tracks:

2
Bitte die Nummer des Tontraegers eingeben:
3
Titel: Kassettentrack2 Laenge: 5 Sekunden

... Menü ...
Auswahl: e

Aufgabe 11.2:

```
INTERFACE Boxer;

TYPE T <: REFANY;  (* Abstrakter Datentyp Spieler *)

PROCEDURE LeseEin(nachricht: TEXT): T;
PROCEDURE Drucke(boxer: T);
PROCEDURE Gleich(spielerA, spielerB: T): BOOLEAN;

END Boxer.
```

```
MODULE Boxer;

IMPORT SIO;  (* Importiere Operationen fuer Ein-/Ausgabe *)
```

```
IMPORT Text; (* Importiere Operationen fuer Textmanipulation
              *)

TYPE Name = RECORD
              vorname : TEXT;
              nachname: TEXT;
            END;

     (* Definiere nach aussen nur als abstrakten Datentyp
        sichtbaren Datentyp fuer Boxer *)

     REVEAL T = BRANDED REF RECORD
                  name       : Name;
                  nation     : TEXT;
                  gewicht    : CARDINAL;
                END;

PROCEDURE LeseEin(nachricht: TEXT): T=
(* Fragt den Benutzer nach den Daten eines Boxers und gibt
   anschliessend einen entsprechenden Boxer zurueck. *)

VAR resultat: T;
    dummy   : TEXT;

BEGIN
  (* Erzeuge einen neuen Boxer auf der Halde *)
  resultat := NEW(T);

  (* Erfrage Daten vom Benutzer *)
  SIO.PutLine(nachricht);
  SIO.PutText("Nachname   : ");
  resultat^.name.nachname := SIO.GetLine();
  SIO.PutText("Vorname    : ");
  resultat^.name.vorname := SIO.GetLine();
  SIO.PutText("Nation     : ");
  resultat^.nation       := SIO.GetLine();
  SIO.PutText("Gewicht: ");
  resultat^.gewicht := SIO.GetInt();

  (* Lies RETURN nach Gewicht aus der Eingabe! *)
  dummy := SIO.GetLine();

  RETURN resultat;
END LeseEin;
```

Entsprechend:
PROCEDURE Drucke(boxer: T)
PROCEDURE Gleich(boxerA, boxerB: T): BOOLEAN

```
BEGIN
END Boxer.
```

```
INTERFACE Boxen;

IMPORT Boxer;  (* Importiere den ADT Boxer *)
IMPORT Assertion AS As;

PROCEDURE VollRL(): BOOLEAN;
PROCEDURE BoxerExistiertRL(boxer: Boxer.T): BOOLEAN;
PROCEDURE EinfuegenBoxerRL(boxer: Boxer.T)RAISES{As.Violated};
(* Fuegt den gegebenen Boxer am Ende der Rangliste ein.
   Vorbedingung: Der Boxer ist noch nicht in der Rangliste
   enthalten und die Rangliste ist noch nicht  voll.
   Vorbedingung: Der einzufuegende Boxer darf noch nicht in
   der Liste enthalten sein.
   Nachbedingung: Der Boxer muss in der Liste enthalten
   sein.*)

PROCEDURE LoescheBoxerRL(boxer:Boxer.T) RAISES{As.Violated};
(* Loescht den gegebenen Boxer aus der Rangliste.
   Vorbedingung:
   Der Boxer ist in der Rangliste enthalten.
   Vorbedingung: Der zu loeschende Boxer muss in der Rangliste
   enthalten sein.
   Nachbedingung: Der zu loeschende Boxer darf nach dem
   Loeschen nicht mehr in der Rangliste enthalten sein. *)

PROCEDURE AktualisiereRL(gewinner, verlierer: Boxer.T)
                                          RAISES{As.Violated};
(* Aendert die Rangfolge in der Rangliste entsprechend dem
   gegebenen Kampfergebnis (Gewinner, Verlierer).
   Vorbedingung: Gewinner und Verlierer sind in der Rangliste
   enthalten
   Vorbedingung: Gewinner und Verlierer muessen beide in der
   Liste enthalten sein. *)

PROCEDURE DruckeRL();

END Boxen.
```

```
MODULE BoxListe EXPORTS Boxen;
(* Dieses Modul implementiert die Verwaltung einer
   Boxrangliste wie in der Schnittstelle Boxen definiert, mit
   Hilfe einer einfach verketteten Liste. *)

IMPORT SIO; (* Importiere Operationen fuer die Ein-/Ausgabe *)
IMPORT Boxer;  (* Importiere den ADT Boxer *)
IMPORT Assertion AS As;

TYPE RanglisteRef = REF RanglistenElement;

     RanglistenElement = RECORD
```

```
                          boxer  : Boxer.T;
                          naechster: RanglisteRef;
                        END;

VAR rangliste    : RanglisteRef; (* Anker der Boxangliste *)
```

Die Implementierung der folgenden Prozeduren sollte klar sein:
PROCEDURE SucheBoxerRL(boxer: Boxer.T): RanglisteRef
(* Sucht den gegebenen Boxer in der Rangliste und gibt einen Zeiger auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der Boxer nicht in der Liste vorhanden ist. *)
PROCEDURE SucheVorgaengerRL(boxer: Boxer.T): RanglisteRef
(* Sucht den Vorgaenger des gegebenen Boxers in der Rangliste und gibt einen Zeiger auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der Boxer keinen Vorgaenger hat, d.h. wenn er das erste Element oder nicht in der Liste ist. *)
PROCEDURE VollRL(): BOOLEAN
PROCEDURE BoxerExistiertRL(boxer: Boxer.T): BOOLEAN

```
PROCEDURE EinfuegenBoxerRL(boxer: Boxer.T)RAISES{As.Violated}=

VAR ranglisteRef, aktRLRef: RanglisteRef;

BEGIN
  (* Vorbedingungen *)
  As.Require(NOT BoxerExistiertRL(boxer), "Der Boxer ist
              schon in der Liste enthalten!");
  As.Require(NOT VollRL(), "Die Liste ist voll!!!");
  (* Erzeuge neues Ranglisten-Element fuer den Boxer *)
  ranglisteRef := NEW(RanglisteRef);
  ranglisteRef^.boxer := boxer;
  ranglisteRef^.naechster := NIL;

  (* Fuege Element an Ende der Liste an *)
  IF (rangliste = NIL) THEN
    (* Liste ist noch leer *)
    rangliste := ranglisteRef;
  ELSE
    (* Suche Ende der Liste *)
    aktRLRef := rangliste;
    WHILE (aktRLRef^.naechster # NIL) DO
      aktRLRef := aktRLRef^.naechster;
    END;
    (* Haenge Boxer ans Ende an *)
    aktRLRef^.naechster := ranglisteRef;
  END;
  (* Nachbedingung *)
  As.Ensure(BoxerExistiertRL(boxer), "Fehler beim Einfuegen
   des Boxers!! Der Boxer ist nicht in der Liste enthalten!");

END EinfuegenBoxerRL;

PROCEDURE LoescheBoxerRL(boxer: Boxer.T) RAISES{As.Violated}=
```

```
VAR ranglisteRef,
    vorgaengerRef: RanglisteRef;

BEGIN
   (* Vorbedingung *)
   As.Require(BoxerExistiertRL(boxer), "Der Boxer ist nicht in
   der Liste enthalten!");
   (* Suche den Boxer in der Liste *)
   ranglisteRef := SucheBoxerRL(boxer);

  IF (ranglisteRef # NIL) THEN

    (* Vorhandenen Boxer aus der Rangliste loeschen *)
    vorgaengerRef := SucheVorgaengerRL(boxer);
    IF vorgaengerRef = NIL THEN
      (* Erstes Element in Rangliste, daher kein Vorgaenger *)
      rangliste := ranglisteRef^.naechster;
    ELSE
      vorgaengerRef^.naechster := ranglisteRef^.naechster;
    END;
  END;
  (* Nachbedingung *)
  As.Ensure(NOT BoxerExistiertRL(boxer), "Fehler beim
              Loeschen! Der Boxer ist noch immer enthalten1");
END LoescheBoxerRL;
```

Entsprechend: PROCEDURE AKommtHinterBInRL(aRef, bRef: RanglisteRef):BOOLEAN

```
PROCEDURE AktualisiereRL(gewinner, verlierer: Boxer.T)
                                          RAISES{As.Violated}=

VAR gewinnerRef, verliererRef      : RanglisteRef;
    vorVerliererRef: RanglisteRef;

BEGIN
  (* Vorbedingungen *)
  As.Require(BoxerExistiertRL(gewinner), "Der Gewinner ist
             nicht in der Liste enthalten!");
  As.Require(BoxerExistiertRL(verlierer), "Der Verlierer ist
             nicht in der Liste enthalten!");

  (* Suche zunaechst Gewinner und Verlierer in der Rangliste
   *)
  gewinnerRef := SucheBoxerRL(gewinner);
  verliererRef := SucheBoxerRL( verlierer);

  (* Pruefe, ob nicht Gewinner ohnehin vor Verlierer in der
     Rangliste! *)
  IF NOT AKommtHinterBInRL(gewinnerRef, verliererRef) THEN
    SIO.PutLine("Keine Veraenderung in der Rangliste!");
  ELSE(* Fuege Verlierer hinter Gewinner in Rangliste ein *)
```

```
      (* Suche die jeweiligen Vorgaenger in der Rangliste *)
      vorVerliererRef := SucheVorgaengerRL(verlierer);

      (* Entferne Verlierer aus ursprünglicher Position *)
      IF ( vorVerliererRef = NIL) THEN
        (* Kein Vorgaenger, da ganz am Anfang der Liste *)
        rangliste := verliererRef^.naechster;
      ELSE
        vorVerliererRef^.naechster := verliererRef^.naechster;
      END;

      (* Hänge Verlierer hinter Gewinner ein *)
      verliererRef^.naechster := gewinnerRef^.naechster;
      gewinnerRef^.naechster := verliererRef;
       (* Nachbedingung *)
      As.Ensure(AKommtHinterBInRL(verliererRef, gewinnerRef),
                "Fehler beim Bearbeiten des Kampfes");
    END;
END AktualisiereRL;
```

Entsprechend: PROCEDURE DruckeRL()

```
BEGIN
  (* Initialisiere Liste mit leerer Rangliste *)
  rangliste := NIL;
END BoxListe.
```

```
MODULE BoxerRl EXPORTS Main;

IMPORT SIO; (* Importiere Operationen fuer die Ein-/Ausgabe *)
IMPORT Boxer; (* Importiere den ADT Boxer *)
IMPORT Boxen; (* Importiere das Objektmodul Boxen *)
IMPORT Assertion AS As;

VAR boxer, gewinner, verlierer  : Boxer.T; (* Variablen des
                                              ADTs Boxer *)
    auswahl                     : CHAR;
    dummy                       : TEXT;
```

Die Implementierung folgender Hilfsprozedur, die Hilfsprozedur, die das Befehlsmenü ausgibt, ist offensichtlich: PROCEDURE DruckeMenue()

```
PROCEDURE BoxerRl() RAISES {As.Terminate}=
  BEGIN
    REPEAT
      TRY
        DruckeMenue();
        (* Frage Benutzer nach seiner Auswahl *)
        SIO.PutText("Auswahl: ");
        auswahl := SIO.GetChar();
        dummy := SIO.GetLine(); (* Lies RETURN aus dem
```

```
                                    Eingabepuffer! *)
        SIO.Nl();

        CASE auswahl OF
        |  'F', 'f' => boxer := Boxer.LeseEin("Boxer
                      Einfuegen"); SIO.Nl();
                      Boxen.EinfuegenBoxerRL(boxer);

        |  'L', 'l' => boxer := Boxer.LeseEin("Boxer
                      Loeschen"); SIO.Nl();
                      Boxen.LoescheBoxerRL(boxer);

        |  'G', 'g' => gewinner := Boxer.LeseEin("Sieger des
                      Kampfes"); SIO.Nl();
                      verlierer := Boxer.LeseEin("Verlierer
                      des Kampfes"); SIO.Nl();
                      Boxen.AktualisiereRL(gewinner,
                      verlierer);

        |  'Z', 'z' => Boxen.DruckeRL();

        |  'E', 'e' => (* Nichts machen, gleich ist ohnehin
                         alles zu Ende! *)
        ELSE
          SIO.PutLine("Ungueltiger Befehl!");
        END (* CASE *)

    EXCEPT
      | As.Violated =>
                SIO.PutLine ("*********** Fehler beim
                              Bearbeiten der Boxrangliste
                              ************");
                RAISE As.Terminate;
      | SIO.Error => SIO.PutLine("SIO-Fehler!!");
    END;

    UNTIL (auswahl = 'e') OR (auswahl = 'E');
  END BoxerRl;

BEGIN
  As.EnableAssertions();
  TRY
     BoxerRl();
  EXCEPT
    |As.Terminate =>
      SIO.PutLine("***  Programm wird beendet wegen des obigen
                   Fehlers ***");
   END;
END BoxerRl.
```

Testlauf

F : Fuege einen Boxer ein
L : Loesche einen Boxer
G : Kampf gemacht
Z : Zeige Rangliste an
E : Exit

Auswahl: f

Boxer Einfuegen
Nachname : Schwer
Vorname : Arno
Nation : D
Gewicht: 200

... Menü ...
Auswahl: f

Boxer Einfuegen
Nachname : Mittel
Vorname : Bert
Nation : Aut
Gewicht: 125

... Menü ...
Auswahl: f

Boxer Einfuegen
Nachname : Leicht
Vorname : Christian
Nation : CH
Gewicht: 70

... Menü ...
Auswahl: z

*** Box-Rangliste ***

1. Schwer Arno Nation D Gewicht 200
2. Mittel Bert Nation Aut Gewicht 125
3. Leicht Christian Nation CH Gewicht 70

... Menü ...

Boxer Loeschen
Nachname : Unbekannt
Vorname : Unbekannt
Nation : Unbekannt
Gewicht: 100

*** Precondition violated: Der Boxer ist nicht in der Liste enthalten!
********** Fehler beim Bearbeiten der Boxrangliste **********
*** Programm wird beendet wegen des obigen Fehlers ***

.... Neu starten mit gleichen Eingaben, wobei die gleichen Boxer wie zuvor in der gleichen Reihenfolge wie zuvor eingegeben werden ...
... Menü ...
Auswahl: z

Sieger des Kampfes
Nachname   : Unbekannt
Vorname   : Unbekannt
Nation   : Unbekannt
Gewicht: 100

Verlierer des Kampfes
Nachname   : a
Vorname   : a
Nation   : a
Gewicht: 1

*** Precondition violated: Der Gewinner ist nicht in der Liste enthalten!
********** Fehler beim Bearbeiten der Boxrangliste **********
*** Programm wird beendet wegen des obigen Fehlers ***

.... Neu starten mit gleichen Eingaben, wobei die gleichen Boxer wie zuvor in der gleichen Reihenfolge wie zuvor eingegeben werden ...
... Menü ...
Auswahl: g

Sieger des Kampfes
Nachname   : a
Vorname   : a
Nation   : a
Gewicht: 1

Verlierer des Kampfes
Nachname   : Unbekannt
Vorname   : Unbekannt
Nation   : Unbekannt
Gewicht: 100

*** Precondition violated: Der Verlierer ist nicht in der Liste enthalten!
********** Fehler beim Bearbeiten der Boxrangliste **********
*** Programm wird beendet wegen des obigen Fehlers ***

Ansonsten, d.h. wenn die Eingaben den Forderungen entsprechen, sehen die Ausgaben wie in der 7. Übung aus.

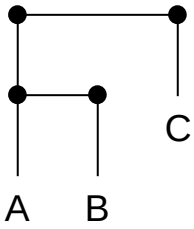
Anmerkung: Die vollständige Implementierung steht im Netz zur Verfügung.

Übung 12
Musterlösung

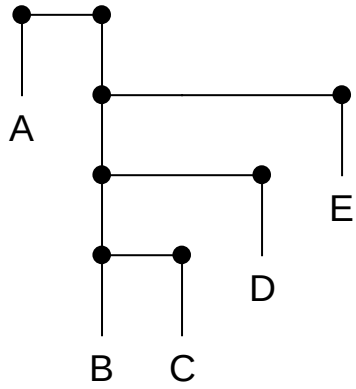
Aufgabe 12.1

a) Speicherstruktur

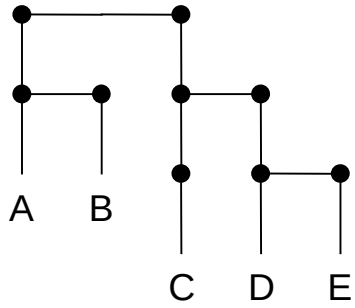
Liste 1



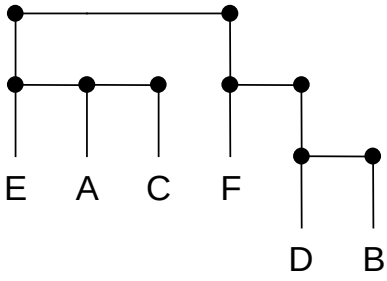
Liste 2



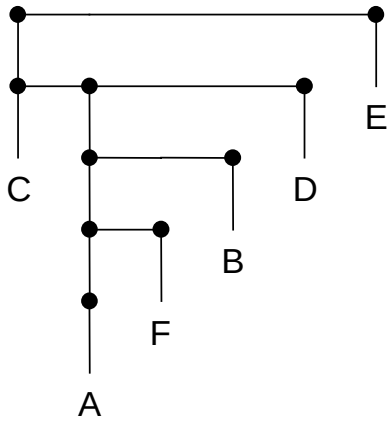
Liste 3



Liste 4



Liste 5



b) Aufbau der Listen aus den Atomen und leeren Listen mittels CONS

```
(setq liste1 (cons (cons 'A (cons 'B ()))
                   (cons 'C ()) ) )

(setq liste2 (cons 'A (cons (cons (cons (cons 'B (cons 'C ()))
                                        (cons 'D ()))
                                  (cons 'E ()))
                            ()) ) )

(setq liste3 (cons (cons 'A (cons 'B ()))
                   (cons (cons (cons 'C ())
                               (cons (cons 'D (cons 'E ()))
                                     ()) )
                         ()) ) )

(setq liste4 (cons (cons 'E (cons 'A (cons 'C ())))
                   (cons (cons 'F (cons (cons 'D (cons 'B ()))
                                        ()) )
                         ()) ) )

(setq liste5 (cons (cons 'C (cons (cons (cons (cons 'A ())
                                              (cons 'F ()))
                                        (cons 'B ()))
                                  (cons 'D ())) )
                   (cons 'E ()) ) )
```

c) Zerlegen der in Teil b) gebauten Liste in die neue Liste (A B C)

```
(setq resultat1 (cons (car (car liste1))
                      (cons (car (cdr (car liste1)))
                            (cdr liste1)) ) )

(setq resultat2 (cons (car liste2)
                      (car (car (car (cdr liste2))))) )

(setq resultat3 (cons (car (car liste3))
                      (cons (car (cdr (car liste3)))
                            (car (car (cdr liste3)))) ) )

(setq resultat4 (cons (car (cdr (car liste4)))
                      (cons (car (cdr (car (cdr (car
                                (cdr liste4))))))
                            (cdr (cdr (car liste4)))) ) )

(setq resultat5 (cons (car (car (car (car (cdr (car liste5))))))
                      (cons (car (cdr (car (cdr (car liste5)))))
                            (cons (car (car liste5))
                                  ()) ) ) )
```

Aufgabe 12.2

a) Funktion `wurzel`:

```
;; Hilfsfunktion
(defun quadrat (x)
    (* x x) )


;; berechnet die Wurzel von a
(defun wurzel (a)

    ; ermittelt Wurzel der Zahl a durch Probieren von 1 aufwaerts
    (defun ermittleWurzel (a zaehler)
        (cond ((< (quadrat zaehler) a)
                  (ermittleWurzel a (+ zaehler 1)))
              ((= (quadrat zaehler) a) zaehler)
              (T 0)) )

    ; starte Probe mit 1
    (ermittleWurzel a 1))
```

Probelauf der Funktionen `quadrat` und `wurzel`:

```
> (quadrat 2)
4
> (quadrat 5)
25
> (wurzel 9)
3
> (wurzel 7)
0
> (wurzel 25)
5
```

b) Prozedur `pythagoras_tripel`:

```
;; die Wurzelfunktion aus Teil a) wird hier verwendet

;; Hilfsfunktion, berechnet die Summe der Quadrate von x und y

(defun quadratsumme (x y)
    (+ (quadrat x) (quadrat y)))
```

```
;; ermittelt durch Probieren aller (u, v)-Kombinationen im
;; Wertebereich (a..b) eine passende Zahl w und gibt bei Erfolg
;; das pythagoraeische Tripel (u, v, w), andernfalls NIL zurueck

(defun pythagoras_tripel (a b)

    ;; Hilfsfunktion, probiert alle Kombinationen von (u, v)
    (defun probiere (u v)
        (cond ((= (wurzel (quadratsumme u v)) 0)
               ; Tripel noch nicht gefunden
               (cond ((and (= u b) (= v b)) ()) ;leere Liste zurueck
                     ((and (< u b) (= v b)) (probiere (+ u 1) a))
                     ((< v b)               (probiere u (+ v 1)))))

              ; passendes Tripel gefunden, als Liste zurueckgeben
              (T (list u v (wurzel (quadratsumme u v))))) )

    ; starte Probe mit u = a und v = a
    (probiere a a))
```

Probelauf der Funktion `pythagoras_tripel`:

```
> (pythagoras_tripel 1 5)
(3 4 5)
> (pythagoras_tripel 3 4)
(3 4 5)
> (pythagoras_tripel 3 5)
(3 4 5)
> (pythagoras_tripel 4 5)
NIL
> (pythagoras_tripel 5 10)
(6 8 10)
> (pythagoras_tripel 10 20)
(12 16 20)
> (pythagoras_tripel 15 16)
NIL
```

Aufgabe 12.3

Prozedur `nimmspiel`:

```
;; das Nimmspiel erwartet als Eingabe die Anzahl der Staebchen
;; zu Beginn des Spiels und spielt anschliessend gegen den
;; Benutzer mittels eines PRINT/READ-Dialogs

(defun nimmspiel (n)
    ; drucke Anzahl der Staebchen im Spiel
    (print (cons n '(Staebchen)))
    ; Benutzer darf das erste Staebchen nehmen!
    (cond ((or (> n 1))
               ; frage Benutzer nach seinem Zug
               (print '(Wieviele Staebchen nehmen Sie ?))
               (let ((n (- n (read))))
                  (print (cons n '(Staebchen)))
                  (cond ((> n 1)
                            (let ((zug (rem (+ n 3) 4)))
                                 (cond ((= zug 0)
                                     (nimmspiel (- n 1)))
                                       ( T
                                     (nimmspiel (- n zug))))))
                        ((= n 1) (print '(Gratuliere !))))))

          ; der Benutzer hat keine Chance mehr!
          ((= n 1)(print '(Ich habe gewonnen !)))))
```

Alternative mit SETQ statt LET:

```
;; Alternative zur obigen Implementierung, die SETQ statt
;; LET verwendet die Funktion arbeitet genauso wie oben,
;; allerdings haben die mit SETQ gesetzten Variablen n und
;; zug nicht nur lokale Gueltigkeit wie bei LET

(defun alt_nimmspiel (n)
    ; drucke Anzahl der Staebchen im Spiel
    (print (cons n '(Staebchen)))
    ; Benutzer darf das erste Staebchen nehmen!
    (cond ((or (> n 1))
               ; frage Benutzer nach seinem Zug
               (print '(Wieviele Staebchen nehmen Sie ?))
               (setq n (- n (read)))
               (print (cons n '(Staebchen)))
               (cond ((> n 1)
                         (setq zug (rem (+ n 3) 4))
                         (cond ((= zug 0) (nimmspiel (- n 1)))
                               (T     (nimmspiel (- n zug)))))
                     ((= n 1) (print '(Gratuliere !)))))

          ; der Benutzer hat keine Chance mehr!
          ((= n 1)(print '(Ich habe gewonnen !)))))
```

Probeläufe der Prozedur `nimmspiel`:

```
> (nimmspiel 15)

(15 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 2

(13 STAEBCHEN)
(12 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 1

(11 STAEBCHEN)
(9 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(6 STAEBCHEN)
(5 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(2 STAEBCHEN)
(1 STAEBCHEN)
(ICH HABE GEWONNEN !)
(ICH HABE GEWONNEN !)
> (nimmspiel 15)

(15 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 2

(13 STAEBCHEN)
(12 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(9 STAEBCHEN)
(8 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(5 STAEBCHEN)
(4 STAEBCHEN)
(WIEVIELE STAEBCHEN NEHMEN SIE ?) 3

(1 STAEBCHEN)
(GRATULIERE !)
(GRATULIERE !)
```

Die Strategie bei diesem Spiel ist, bei jedem Zug so viele Stäbchen zu nehmen, dass die verbleibende Anzahl Stäbchen eins größer ist als die nächste durch vier teilbare Zahl. Gelingt dem Spieler ein solcher Zug, dann hat er das Spiel bereits gewonnen. Das Problem bei diesem Spiel ist, dass wenn auch der Gegner diese Strategie kennt, immer derjenige gewinnt, der die Strategie zuerst umsetzen kann.

Programmierung **WS 2000/2001** **Prof. H. Lichter**
Th. von der Maßen

Übung 13
Musterlösung

Aufgabe 13.1

a)
Der Fehler der Funktion *mymapcar* liegt darin, daß sie die Antwortliste in falscher Reihenfolge liefert. In jedem Durchlauf der Funktion *iter* wird an die bestehende, anfänglich leere Antwortliste, das jeweils erste und nun mit `f` bearbeitete Element vorne angehängt.:
`(cons (funcall f (car liste)) ergebnisliste)`.

b)
```
(defun mymapcar2 (f liste)
  (defun iter (liste ergebnisliste)
    (cond ((null liste) (reverse(ergebnisliste)))
          (t (iter (cdr liste)
                   (cons (funcall f (car liste)) ergebnisliste)))))
  (iter liste ()))
```

c)
```
(mymapcar2 (lambda (x) (* x x)) '(1 2 3 4))
```
Ergebnis : `(1 4 9 16)`

Aufgabe 13.2

a)
```
studentHoertVorlesung(balin, kryptographie).
studentHoertVorlesung(beutlin, compilerbau).
studentHoertVorlesung(boffin, softwaretechnik).
studentHoertVorlesung(brandybock, mustererkennung).
studentHoertVorlesung(pausbacken, mustererkennung).
studentHoertVorlesung(straffguertel, compilerbau).
studentHoertVorlesung(bolger, datenkommunikation).
studentHoertVorlesung(honblaeser, softwaretechnik).
studentHoertVorlesung(stolzfuss, datenkommunikation).

profLiestVorlesung(hromkovic, kryptographie).
profLiestVorlesung(indermark, compilerbau).
profLiestVorlesung(nagl, softwaretechnik).
profLiestVorlesung(spaniol, datenkommunikation).
profLiestVorlesung(ney, mustererkennung).
```

b)
```
balin_hoert_bei_hromkovic :- studentHoertVorlesung(balin, kryptographie),
                             profLiestVorlesung(hromkovic, kryptographie).

student_hoert_bei_nagl(Student) :- studentHoertVorlesung(Student,
                                   softwaretechnik),
                                   profLiestVorlesung(nagl, softwaretechnik).
```

```
student_hoert_bei(Student, Prof) :- studentHoertVorlesung(Student, Vorlesung),
                                    profLiestVorlesung(Prof, Vorlesung).


beutlin_ist_kommilitoneVon_balin :- studentHoertVorlesung(beutlin, Vorlesung),
                                    studentHoertVorlesung(balin, Vorlesung).

kommilitoneVon(Stud1, Stud2) :- studentHoertVorlesung(Stud1, Vorlesung),
                                studentHoertVorlesung(Stud2, Vorlesung),
                                not(Stud1=Stud2).
```

c)

```
?- studentHoertVorlesung(balin, compilerbau).
NO

?- studentHoertVorlesung(balin, kryptographie).
YES

?- profLiestVorlesung(nagl, softwaretechnik).
YES

?- profLiestVorlesung(ney, compilerbau).
NO

?- studentHoertVorlesung(brandybock, V).
V = mustererkennung
NO

?- profLiestVorlesung(spaniol, V).
V = datenkommunikation
NO

?- profLiestVorlesung(P, mustererkennung).
P = ney
NO

?- studentHoertVorlesung(S, softwaretechnik).
S = boffin  ;
S = hornblaeser ;
NO

?- student_hoert_bei(S, indermark).
S = beutlin  ;
S = straffguertel  ;
NO

?- student_hoert_bei(S, P).
S = balin
P = hromkovic  ;

S = beutlin
P = indermark  ;

S = boffin
P = nagl  ;

S = bandybock
P = ney  ;

S = pausbacken
P = ney  ;
```

```
S = straffguertel
P = indermark  ;
S = bolger
P = spaniol  ;

S = hornblaeser
P = nagl  ;

S = stlzfuss
P = spaniol  ;

NO


?- beutlin_ist_kommilitoneVon_balin.
NO

?- kommilitoneVon(beutlin, straffguertel).
YES

?- kommilitoneVon(bolger, K).
K = stolzfuss  ;
NO

?- kommilitoneVon(K1, K2).
K1 = beutlin
K2 = straffguertel  ;

K1 = boffin
K2 = hornblaeser  ;

K1 = brandybock
K2 = pausbacken  ;
K1 = pausbacken
K2 = brandybock  ;

K1 = straffguertel
K2 = beutlin  ;

K1 = bolger
K2 = stolzfuss  ;

K1 = hornblaeser
K2 = boffin  ;

K1 = stolzfuss
K2 = bolger  ;

NO
```

Aufgabe 13.3

a)

```
farbe(rot).
farbe(gelb).
farbe(blau).
farbe(gruen).

verschieden(rot, gelb).
verschieden(rot, gruen).
verschieden(rot, blau).
verschieden(gelb, rot).
verschieden(gelb, gruen).
verschieden(gelb, blau).
verschieden(gruen, rot).
verschieden(gruen, gelb).
verschieden(gruen, blau).
verschieden(blau, gelb).
verschieden(blau, rot).
verschieden(blau, gruen).

korrekteFaerbung(L1, L2, L3, L4, L5, L6, L7, L8) :-
                  verschieden(L1, L2), verschieden(L1, L3),
                  verschieden(L1, L4), verschieden(L1, L5),
       verschieden(L2, L5), verschieden(L2, L6), verschieden(L2, L7),
                  verschieden(L3, L4), verschieden(L3, L5),
                  verschieden(L3, L6), verschieden(L4, L5),
                  verschieden(L5, L6), verschieden(L5, L7),
                  verschieden(L6, L7), verschieden(L7, L8).
```

b)

Eine Lösung des Problems erhalten wir, wenn wir nach einer Färbung der acht Länder fragen, bei der nur die vier erlaubten Farben benutzt werden und die auch korrekt ist:

```
?- farbe(L1), farbe(L2), farbe(L3), farbe(L4), farbe(L5), farbe(L6),
   farbe(L8), korrekteFaerbung(L1, L2, L3, L4, L5, L6, L7, L8).
```

Eine korrekte Färbung der Landkarte lautet:
`L1=rot, L2=gelb, L3=gelb, L4=blau, L5=gruen, L6=rot, L7=blau, L8=rot`

c)

Insgesamt gibt es 360 Lösungen für eine korrekte Färbung der Landkarte.

Programmierung WS 2000/01 **Prof. H. Lichter**
A. Nowack

Musterlösung
Testklausur

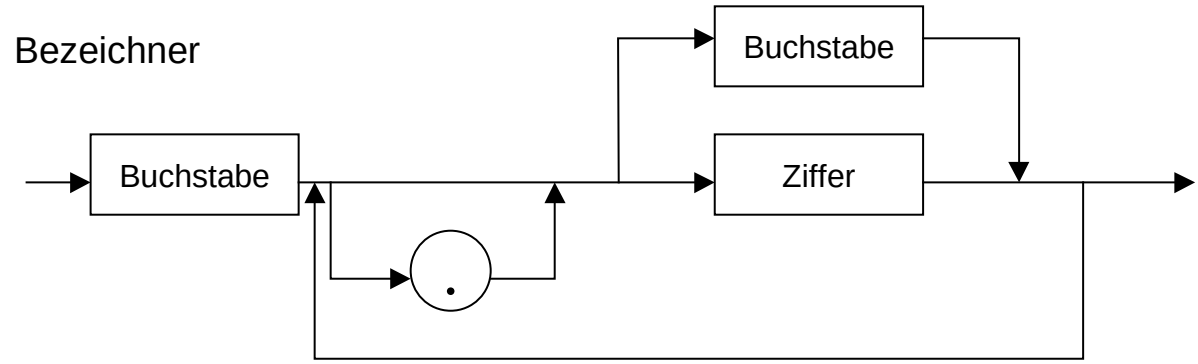
Aufgabe 1 : Syntaxdiagramm und EBNF

Es sollen Bezeichner definiert werden, die folgenden Vorschriften genügen:

- Bezeichner bestehen aus Buchstaben, Ziffern und Punkten.
- Jeder Bezeichner beginnt mit einem Buchstaben.
- Die Bezeichner sind beliebig lang, aber nicht kürzer als zwei Zeichen
- Bezeichner dürfen durch Punkte gegliedert werden, jedoch dürfen diese weder am Ende noch mehrfach hintereinander stehen (d.h. „ABC.“ Und „AB..C“ sind falsch).

Die Produktionen für **Buchstabe** und **Ziffer** können als gegeben betrachtet werden.

1.1 Geben Sie die Syntax für Bezeichner durch ein Syntaxdiagramm an!
Die Lösung muß knapp und eindeutig sein.

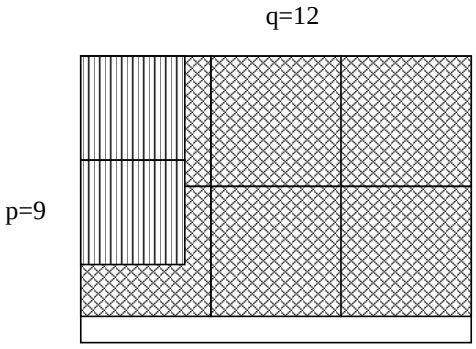


1.2 Wie verändert sich die Lösung, wenn zusätzlich gelten soll, daß ein Bezeichner höchstens eine Ziffer enthalten darf?
Formulieren Sie die veränderte Syntaxbeschreibung in EBNF!

```
Bezeichner      = Buchstabe Rest
Rest            = ["."] ( Buchstabe  [Rest] | Ziffer { ["." ] Buchstabe } )
```

Aufgabe 2: Kacheln

Bei der folgenden Aufgabe sind alle Längenmaße Vielfache der Grundeinheit E = 10 cm. Die Gewichte (präzise: Massen) sind Vielfache der Masse einer kleinen Kachel (10 cm im Quadrat), die m = 100 g wiegt. Die Kachel mit doppelter Kantenlänge ist entsprechend viermal so schwer usw.
Es stehen quadratische Kacheln aller Größenstufen bis zur Kantenlänge kmax*E zur Verfügung, also mit den Kantenlängen E, 2E, ... , kmax*E.
Ein Rechteck der Länge p*E und der Breite q*E (p, q $\in$ N) ist mit den größten verfügbaren und in das Rechteck passenden Kacheln zu belegen, wobei diese nicht über das Rechteck hinausragen dürfen. Sie bilden nun ein neues Rechteck, das mit den nächstkleineren Kacheln ausgelegt wird usw., bis schließlich in der obersten Schicht Kacheln der Größe E liegen.
Für ein gegebenes Tripel p, q, kmax ist das Gesamtgewicht der Kacheln zu berechnen, die auf dem Rechteck liegen. Beispiel: p = 9, q = 12, kmax = 4.



Die Abbildung zeigt, daß 2*3=6 Kacheln der Größe 4 auf das Rechteck passen, darauf 2*4=8 Kacheln der Größe 3 (von denen zwei links gezeigt sind). Auf diese wiederum kommen 3*6 der Größe 2 und 6*12 der Größe 1. Das ergibt ein Gesamtgewicht von (6*16 + 8*9 + 18*4 + 72*1) * 100 g = 31 200 g.

2.1 Geben Sie eine Modula-3 Funktion an, die aus den Parametern p, q, und kmax das Gewicht (in g) der Kacheln nach obigem Berechnungsschema liefert.

```
PROCEDURE BerechneKacheln (p, q, kmax : INTEGER): INTEGER =
VAR panz, qanz : INTEGER;
BEGIN
  IF kmax > 0 THEN
    qanz := q DIV kmax; panz := p DIV kmax;

    IF (panz > 0) AND (qanz > 0) THEN

      (* Kacheln der Breite bzw. Höhe kmax können verwendet werden *)
      SIO.PutInt((qanz * panz) * (kmax *kmax)); SIO.Nl();

      (* Masse dieser Schicht + Masse des rekursiv berechneten Rests = Ergebnis *)
      RETURN (((qanz * panz) * (kmax *kmax) * 100) +
                  BerechneKacheln(panz * kmax, qanz * kmax, kmax - 1));
    ELSE
      (* Probiere nächst kleinere Kacheln! *)
      RETURN BerechneKacheln (p, q, kmax - 1);
    END;
  ELSE
    RETURN 0;
  END;
END BerechneKacheln;
```

Aufgabe 3: Programmanalyse

Gegeben sei die folgende rekursive Prozedur:

```
PROCEDURE Unknown (p1 : INTEGER; p2 : BOOLEAN) =
BEGIN
  IF NOT p2 THEN
    SIO.PutInt (p1 DIV 7);
    SIO.Nl();
  END;
  IF p2 OR (p1 > 10) THEN
    Unknown ( (p1 + 33) MOD 41, p2 # (p1 < 20))
  END;
END Unknown;
```

Die Prozedur erzeugt also mit jeder Inkarnation keine oder ein Zeile Augabe.

3.1 Geben Sie nach dem Muster des Beispiels (Aufruf Unknown (10, TRUE)) die Inkarnationen von Unknown an, die aus dem Aufruf Unknown (30, FALSE) entstehen, und zwar bis zum Abbruch der Rekursion oder bis zur letzten Zeile in der dafür vorgesehenen Tabelle.

#	p1	p2	Ausgabe (falls erzeugt)
1	10	TRUE	-
2	2	FALSE	0 Abbruch

#	p1	p2	Ausgabe (falls erzeugt)
1	30	FALSE	4
2	22	FALSE	3
3	14	FALSE	2
4	6	TRUE	-
5	39	FALSE	5
6	31	FALSE	4
7	23	FALSE	3

Aufgabe 4: Listen

Seien $x = (x_1,x_2,\ldots,x_m)$ und $y = (y_1,y_2, \ldots, y_n)$ zwei nicht-leere einfach verkettete Listen.

Aus diesen Listen soll kann eine neue Liste Z konstruiert werden, die folgenden Bedingungen genügt.

$z = (x_1,y_1, x_2,y_2,\ldots,x_m,y_m,y_{m+1}, \ldots, y_n)$	falls n > m
$z = (x_1,y_1, x_2,y_2,\ldots,x_n,y_n,x_{n+1}, \ldots, x_m)$	falls m > n
$z = (x_1,y_1, x_2,y_2,\ldots,x_m,y_n)$	falls n = m

4.1 Entwerfen Sie eine geeignete Datenstruktur, und schreiben Sie eine Modula-3 Prozedur `ErzeugeZ`, die aus zwei nicht-leeren einfach verketteten Listen x und y eine neue Liste z gemäß obiger Spezifikation erzeugt (die beiden Eingabe-Listen werden dabei zerstört)

Datenstruktur:
```
TYPE Liste = REF ListenElement;
     ListenElement = RECORD
                       inhalt : INTEGER;
                       naechster : Liste;
                     END;
```

Prozedur:

```
PROCEDURE MischeListen( VAR x,y,z : Liste)=
VAR hz : Liste;

BEGIN
  z := x;
  x := x^.nachfolger;
  z^.nachfolger := y;
  y := y^.nachfolger;
  hz := z^.nachfolger;

 (* Durchlaufen der Listen, bis das Ende einer Liste erreicht ist *)
  WHILE x # NIL AND y # NIL DO
    hz^.nachfolger := x;
    x := x^.nachfolger;
    hz := hz^.nachfolger;
    hz^.nachfolger := y;
    y := y^.nachfolger;
    hz := hz^.nachfolger;
  END;

  (* Hänge Rest der Liste an, deren Ende noch nicht erreicht wurde *)
  IF x = NIL THEN
   hz^.nachfolger := y;
  ELSE
   hz^.nachfolger := x;
  END;
END MischeListen;
```

Die beiden Listen werden (am Anfang beginnend) durchlaufen, bis bei einer der beiden Listen das Ende erreicht ist. Während des Durchlaufens werden abwechselnd Elemente aus den beiden Listen in die Ergebnisliste eingefügt. Ist das Ende der einen Liste erreicht, das Ende der anderen jedoch noch nicht, so wird der Rest der Liste, deren Ende noch nicht erreicht ist, an das Ende der Ergebnisliste angehängt.

Aufgabe 5: Abstrakter Datentyp

Die Realisierung von Mengen in Modula-3 ist beschränkt auf Ordinaltypen als Elementtyp der Menge. Sollen andere Elementtypen in Mengen verwendet werden, so muß dafür eine geeignete Implementierung erstellt werden.

Entwerfen Sie einen Abstrakten Datentyp SetOfText, der eine Menge realisiert, deren Elementtyp der vordefinierte Typ TEXT ist. Als Operationen sollen bereitgestellt werden: Vereinigung, Schnitt, Aufnahme und Entfernen eines Elements aus der Menge sowie der Test des Enthaltenseins.

5.1 Geben Sie das Interface-Modul für den ADT SetOfText an!

```
INTERFACE SetOfTextADT;

TYPE SetOfText <: REFANY;

PROCEDURE Erzeuge () : SetOfText;
PROCEDURE Vereinigung (s1,s2 : SetOfText) : SetOfText;
PROCEDURE Schnitt (s1,s2 : SetOfText) : SetOfText;
PROCEDURE Aufnehmen (VAR s : SetOfText; t : TEXT);
PROCEDURE Entfernen (VAR s : SetOfText; t : TEXT);
PROCEDURE Enthaelt (s: SetOfText; t : TEXT) : BOOLEAN;


END SetOfTextADT.
```

5.2 Geben Sie den Deklarationsteil des Implementierungs-Moduls des ADTs `SetOfText` an (d.h. alles bis zur ersten implementierten Funktion).
Dabei sollen Mengen prinzipiell unendlich viele Elemente aufnehmen können.

```
REVEAL SetOfText = BRANDED REF RECORD
                     t : TEXT;
                     naechstesElement : SetOfText;
                END;
```

5.3 Implementieren Sie die Funktion der Mengenvereinigung des ADTs `SetOfText` gemäß Ihrer Deklaration im Interface-Modul!

Die angegebene Prozedur arbeitet wie folgt:
Die “Menge“ resSet soll am Ende das Ergebnis der Vereinigung enthalten.
Zunächst wird resSet mit der “leeren Menge“ initialisiert.
Dann werden nacheinander alle Elemente von s1 in die “Menge“ resSet eingefügt, danach diejenigen aus s2.
Das Einfügen geschieht mit Hilfe der Prozedur “Aufnehmen“. Bei der Verwendung von “Aufnehmen“ wird wird vorausgesetzt, daß ein Element, das in der “Menge“ bereits vorhanden ist, nicht noch einmal eingefügt wird. Somit müssen Elemente, die in beiden Mengen auftreten, nicht durch einen Sonderfall behandelt werden.

```
PROCEDURE Vereinigung (s1, s2 : SetOfText) : SetOfText =
VAR resSet : SetOfText;

BEGIN
  (* Initialisieren mit der leeren Menge *)
  resSet := Erzeuge();

  (* Einfügen der Element von s1 *)
  WHILE s1 # NIL DO
    Aufnehmen(resSet, s1.t);
    s1 := s1^.naechstesElement;
  END;

  (* Einfügen der Elemente von s2 *)
  WHILE s2 # NIL DO
    Aufnehmen(resSet, s2.t);
    s2 := s2^.naechstesElement;
  END;
  RETURN resSet;
END Vereinigung;
```

Aufgabe 6: Klassenhierarchie und Objekttypen

Sie haben die Aufgabe, ein Programm zu entwickeln, mit dem Literaturreferenzen verwaltet werden sollen. Dabei stellen Sie fest, daß es die folgenden vier verschiedenen Arten von Literaturreferenzen gibt.

Referenz auf ein Buch : diese ist charakterisiert durch Angabe von Autor, Titel, Erscheinungsjahr und Verlag

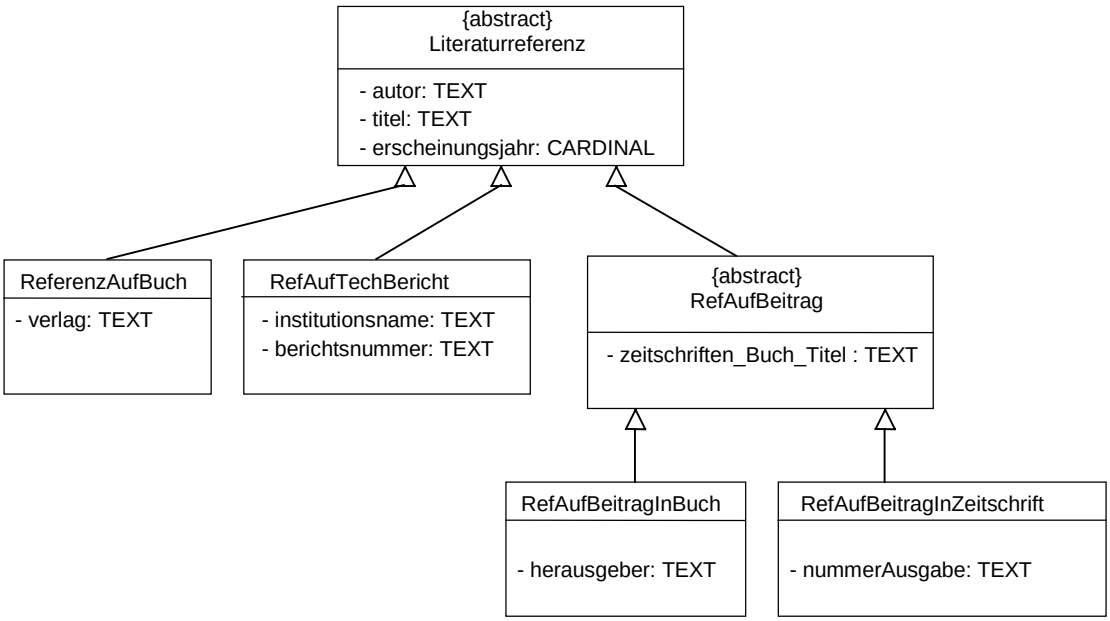
Referenz auf einen Technischen Bericht: diese ist charakterisiert durch Angabe von Autor, Titel, Erscheinungsjahr, Name der Institution und Nummer des Berichts.

Referenz auf einen Beitrag in einem Buch: diese ist charakterisiert durch Angabe von Autor, Titel des Beitrags, Erscheinungsjahr, Buchtitel und Herausgeber.

Referenz auf einen Beitrag in einer Zeitschrift: diese ist charakterisiert durch Angabe von Autor, Titel des Beitrags, Erscheinungsjahr, Zeitschriftentitel und Ausgabennummer.

6.1 Entwerfen Sie eine geeignete Klassenhierarchie, um die oben aufgelisteten Arten an Literaturreferenzen zu realisieren.

Achten Sie dabei darauf, daß gemeinsame Merkmale in abstrakten Klassen zusammen-gezogen werden. Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation (geben Sie lediglich pro Klasse den Namen der Klassen und die Bezeichner der Exemplarvariablen an).



6.2 Geben Sie die Deklaration der Objekttypen (keine geschützten Objekttypen) für die Klassen `ReferenzAufBuch` und `ReferenzAufBeitragInZeitschrift` an. Dabei soll berücksichtigt werden, daß alle Objekte der Klassenhierarchie die Nachrichten `initialisiere` (initialisiert alle Merkmale einer Literaturreferenz mit Standard-Werten) und `alsText` (liefert einen TEXT zurück, der die textuelle Repräsentation aller Merkmale einer Literaturreferenz enthält) kennen müssen. Geben Sie für die beiden Klassen (Objekttypen) auch die notwendigen Zugriffsmethoden für deren spezielle Exemplarvariablen an.

Hinweis: Verwenden Sie der Einfachheit halber ausschließlich TEXT als Typ für die Exemplarvariablen der Klassen.

```
TYPE ReferenzAufBuch =
       LiteraturRef OBJECT
         verlag: TEXT;

       METHODS
         setzeVerlag(name: TEXT) := SetzeVerlag;
         gibVerlag(): TEXT       := GibVerlag;

       OVERRIDES
         initialisiere() := Inititialisiere;
         alsText(): TEXT := AlsText;

       END;


     ReferenzAufBeitragInZeitschrift =
       ReferenzAufBeitrag OBJECT
         nummerAusgabe     : TEXT;

       METHODS
         setzeNummerAusgabe(nr: TEXT) := SetzeNummerAusgabe;
         gibNummerAusgabe(): TEXT     := GibNummerAusgabe;

       OVERRIDES
         initialisiere() := Initialisiere;
         alsText(): TEXT := AlsText;
       END;
```

6.3 Implementieren Sie die zur Nachricht `alsText` gehörende Prozedur der Klasse `ReferenzAufBeitragInZeitschrift.`

```
PROCEDURE AlsText(self: ReferenzAufBeitragInZeitschrift): TEXT=
BEGIN
  RETURN ReferenzAufBeitrag.alsText() & ", " &
         self.gibNummerAusgabe();
END AlsText;
```

Aufgabe 7: Lisp

In dieser Aufgabe seien Mengen von Zahlen identisch mit aufsteigend geordneten Listen von Zahlen.
Z. B. entspricht hierbei die Menge {8, 3, 6, 2} der Liste (2, 3, 6, 8) und die Menge {9, 7, 1, 4, 3} der Liste (1, 3, 4, 7, 9). Die Vereinigung der beiden Mengen entspricht der Liste (1, 2, 3, 4, 6, 7, 8, 9).

7.1 Implementieren Sie in LISP eine Funktion `vereinigungs-menge`, die die Vereinigung zweier solcher Mengen berechnet.

```
(defun vereinigungs-menge (m1 m2)

;; wenn eine der beiden Mengen leer ist, dann ist die Vereinigung
;; gleich der anderen Menge

  (cond ((null m1) m2)
        ((null m2) m1)
;; lokale Variablen für die ersten Elemente der Listen

        (T (let ((x1 (car m1)) (x2 (car m2)))

;; Die Vereinigung hängt von dem Verhältnis der ersten Elemente
;; zueinander ab
;; Bei Gleichheit darf das Element nur einmal in der die Vereinigung
;; repräsentierenden Liste auftreten. Deren erstes Element ist x1
;; bzw. x2 und der Rest ergibt aus der Vereinigung der beiden Reste

            (cond ((= x1 x2) (cons x1 (vereinigungs-menge (cdr m1)
                                                          (cdr m2))))

;; Ist x1 kleiner als x2, dann sind alle Elemente der zweiten Liste
;; größer als die der ersten. Somit auch die der die Vereingung
;; repräsentierenden Elemente. Somit ist x1 das erste Element der
;; Ergebnisliste.

                  ((< x1 x2) (cons x1 (vereinigungs-menge (cdr m1)
                                                          m2)))

;; Es bleibt noch der Fall (< x2 x1) übrig. Dieser ist symmetrisch
;; zum vorhergehenden Fall.

                  (T (cons x2 (vereinigungs-menge m1 (cdr m2)))))))) )
```

7.2 Schätzen Sie den zeitlichen Aufwand Ihrer Funktion für die Berechnung der Vereinigungsmenge ab.
Ist der Aufwand Ihrer Funktion nicht in *O(n)*, so überlegen Sie, wie sich eine Funktion schreiben läßt, die die Vereinigung in der Zeit *O(n)* berechnet.

Mit jedem Rekursionsschritt wird das Problem der Mengenvereingung auf die Vereinigung echt kleinerer Mengen reduziert, bis an mindestens einer Stelle die leere Menge erreicht ist. Genauer wird aus mindestens einer Menge ein Element eliminiert. Somit ist die Anzahl der (Rekursions-)Schritte höchstens gleich der Summe der Anzahl der Elemente der Mengen, also |m1|+|m2| (genauere Abschätzung: min{|m1|, |m2|}+1).
Daraus folgt, daß die Vereinigung in der Zeit O(n) berechnet wird.
Überlegungen zur Optimierung sind also nicht notwendig.