Vorname	Name	Fach	Sem.	Matr.-Nr

Hinweise zur Bearbeitung:

- Schreiben Sie auf jedes Blatt **Vorname**, **Name, Fach, Semester** und **Matrikelnummer**
- Beantworten Sie die Fragen lesbar und auch **im Detail** korrekt.
- Bitte beantworten Sie die Aufgaben auf den **Aufgabenblättern** (benutzen Sie auch die Rückseiten). Antworten auf anderen Blättern können nur berücksichtigt werden, wenn **Name, Matrikelnummer und Aufgabennummer** deutlich darauf erkennbar ist
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch **Durchstreichen**.
- Werden Täuschungsversuche beobachtet, so wird die Klausur mit nicht bestanden bewertet.
- Geben Sie am Ende der Klausur alle Blätter zusammen mit den Aufgabenblättern ab.

Die Organisatoren wünschen allen Kandidaten viel Erfolg!

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**10**	
Aufgabe 2	**8**	
Aufgabe 3	**10**	
Aufgabe 4	**8**	
Aufgabe 5	**10**	
Aufgabe 6	**10**	
Aufgabe 7	**14**	
SUMME	**70**	

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 1 : EBNF

Eine Sprache L soll die folgenden Eigenschaften erfüllen:

1. Alle Wörter der Sprache bestehen aus den Zeichen "X", "O" und "+". Nicht alle drei Zeichen müssen in einem Wort vorkommen, andere sind unzulässig.
2. Jedes Wort der Sprache besteht aus mindestens zwei Zeichen. Die Wörter können beliebig lang sein.
3. "+" steht nicht am Anfang oder am Ende eines Wortes, auch im Wort nicht mehrfach direkt hintereinander.
4. Die Wörter der Sprache enthalten Teilwörter, die folgendermaßen definiert sind. Ein Teilwort bezeichnet den Abschnitt eines Wortes, der kein Zeichen "+" enthält, aber daran grenzt (oder am Anfang oder am Ende des Wortes steht). Beispielsweise bestehen Wa und Wb nur aus einem Teilwort, Wc enthält die Teilwörter "O", "OXO" und "O". Allgemein gilt, daß jedes Teilwort höchstens ein "X" und mindestens ein "O" enthält.

Nach diesen Regeln gehören Wa bis Wd zur Sprachen L, We bis Wh gehören nicht zur Sprache.

Wa = OXOO
Wb = OOOOOOX
Wc = O+OXO+O
Wd = XOOOOO+O+OOXO+OX

We = X=+1
Wf = X
Wg = +O+O+
Wh = XXO+X

1.1 (10 Pkt.) Geben Sie eine möglichst einfache Syntaxdefinition von L in EBNF an. L soll durch Ihre Definition nicht weiter eingeschränkt sein, als die Vorschriften 1 bis 4 erfordern.

Teilwort = { "O"} ("O" "X" | "X" "O" | "O" "O") {"O"}.

L = (Teilwort | "O" "+" (Teilwort | "O") { "+" (Teilwort | "O")}

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 2: Matrixrechnung

Gegeben sei eine (m,n) Matrix M und ein Vektor B mit n Elementen. Der Vektor B und die Matrix M sollen miteinander multipliziert werden.

Hinweis: Die Multiplikation eines Zeilenvektors A =(a1 a2 .. an) der Matrix M mit dem Vektor B = (b1 b2 .. bn) ist folgendermaßen definiert

$$A * B = \sum_{i=1}^{n} a_i * b_i$$

2.1 (2 Pkt.) Geben Sie die Deklarationen für die Typen Matrix, Vektor und Ergebnisvektor an!

```
TYPE IndexM =: [1 .. M];
     IndexN = [1 .. N];

     Matrix = ARRAY IndexM, IndexN OF INTEGER;
     Vektor = ARRAY IndexN OF INTEGER;
     Ergebnisvektor = ARRAY IndexM OF INTEGER;
```

2.2 (6 Pkt.) Schreiben Sie eine Modula-3 Funktion, die eine (m,n) Matrix und einen Vektor mit n Elementen als Eingabe erhält und den Ergebnisvektor der Multiplikation von Matrix und Vektor zurückgibt.

```
PROCEDURE Multipliziere (m : Matrix, b : Vektor ) : Ergebnisvektor =
VAR ergv : Ergebnisvektor:
    wert : INTEGER;
BEGIN
  FOR i := 1 TO M DO
    wert := 0;
    FOR j := 1 TO N DO
      wert := wert + m[i,j] * b[j];
    END;
    ergv[I] := wert;
  END;
  RETURN ergv;
END Multipliziere;
```

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 3: Summe und Produkt

3.1 (5 Pkt.) Schreiben Sie eine rekursive Modula-3 Funktion SUMME, die die Summe zweier CARDINAL Zahlen ermittelt. Dabei dürfen bis auf die vordefinierten Funktionen INC und DEC keine weiteren Standard-Operatoren (+,-, …) verwendet werden.

```
PROCEDURE Summe (a,b : CARDINAL) : CARDINAL =
VAR erg : CARDINAL;
BEGIN
  IF b > 0 THEN
    INC(a); DEC(b);
    erg := Summe (a, b);
  ELSE
    erg := a;
  END;
  RETURN erg;
END Summe;
```

3.2 (5 Pkt.) Schreiben Sie eine rekursive Modula-3 Funktion PRODUKT, die das Produkt zweier CARDINAL Zahlen ermittelt und dazu die von Ihnen definierte Funktion SUMME verwendet. Es dürfen bis auf die vordefinierten Funktionen INC und DEC wiederum keine weiteren Standard-Operatoren (+,-, …) verwendet werden.

```
PROCEDURE Produkt (a,b : CARDINAL) : CARDINAL =
VAR erg : CARDINAL;
BEGIN
  IF b > 0 THEN
    DEC(b);
    erg := Summe (Produkt(a, b), a);
  ELSE
    erg := 0;
  END;
  RETURN erg;
END Produkt;
```

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 4: 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.

4.1 (8 Pkt.) 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

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 5: Vermehrung

Auf einem unbekannten Planeten leben Wesen, die sich von den uns bekannten Lebewesen erheblich unterscheiden. Sie werden im Winter geboren und leben dann maximal 5 Jahre. Im Sommer verbinden sie sich, soweit möglich, in Dreiergruppen, die sich im Winter wieder aufspalten, wobei ein neues Lebewesen entsteht (d.h. aus drei alten Lebewesen sind drei alte plus ein neues Lebewesen entstanden).

Wir nehmen an, daß im Winter des Jahres 0 nur drei neugeborene Lebewesen vorhanden sind. Wieviel Lebewesen können nach Ablauf von n Jahren im Frühling vorhanden sein, wenn man annimmt, daß alle Lebewesen die volle Lebensdauer erreichen und sich maximal vermehren.

5.1 (10 Pkt.) Schreiben Sie eine Modula-3 Funktion `AnzahlLebewesen`, die n als Parameter erhält und die gesuchte Zahl liefert.

Hinweis: Dazu ist es sinnvoll, eine eigene Funktion zu realisieren, die die neugeborenen Lebewesen eines Jahres bestimmt.

```
PROCEDURE Lebewesen(jahr : INTEGER): INTEGER =
VAR anzahl : INTEGER;
BEGIN
  IF jahr < 0 THEN
    RETURN 0;
  ELSIF jahr = 0 THEN
    RETURN 3;
  ELSE
    anzahl := Lebewesen(jahr-1) - NeugeborenLebewesen(jahr-6);
    RETURN anzahl + (anzahl DIV 3);
  END;
END Lebewesen;

PROCEDURE NeugeborenLebewesen(jahr : INTEGER): INTEGER =
VAR anzahl : INTEGER;
BEGIN
  IF jahr < 0 THEN
    RETURN 0;
  ELSIF jahr = 0 THEN
    RETURN 3;
  ELSE
    anzahl := Lebewesen(jahr-1) - NeugeborenLebewesen(jahr-6);
    RETURN anzahl DIV 3;
  END;
END NeugeborenLebewesen;
```

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 6: Listen

Sie haben eine nichtleere, doppelt verkette zyklische Liste mit den Deklarationen

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

Es ist möglich, daß die Verkettung beschädigt ist, daß also beim Durchlaufen der Vorwärts- und der Rückwärtsverkettung scheinbar verschiedene Ketten sichtbar werden. Beispiel:

Wenn vorwärts die Elemente	1, 2, 3, 4, 5, 6, 7, 1 erreicht werden
dann müßten rückwärts	1, 7, 6, 5, 4, 3, 2, 1 erreicht werden;
also nicht	1; 7, 6, 4, 2, 1

Hinweis: Es ist dabei immer sichergestellt, daß keine Zeiger ins Leere weisen und daß das erste Element immer erreicht werden kann.

6.1 (10 Pkt.) Schreiben Sie eine Funktion `Text`, die die die Kette prüft und TRUE liefert, wenn die Verkettung in Ordnung ist, sonst FALSE.

```
PROCEDURE Test(kette: Link): BOOLEAN=
VAR vorElement, nachElement: Link;
    verkettungOK           : BOOLEAN;
BEGIN
  nachElement := kette;
  vorElement  := kette^.vor;  (* Nicht leere Kette! *)

  verkettungOK := TRUE;
  REPEAT
    IF (vorElement = NIL) OR
       (nachElement # vorElement^.nach) THEN
      verkettungOK := FALSE;
    ELSE
      nachElement := vorElement;
      vorElement := vorElement^.vor;
    END;
  UNTIL (vorElement = kette^.vor) OR NOT verkettungOK;

  RETURN verkettungOK;
END Test;
```

Vorname	Name	Fach	Sem.	Matr.-Nr

Aufgabe 7: 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

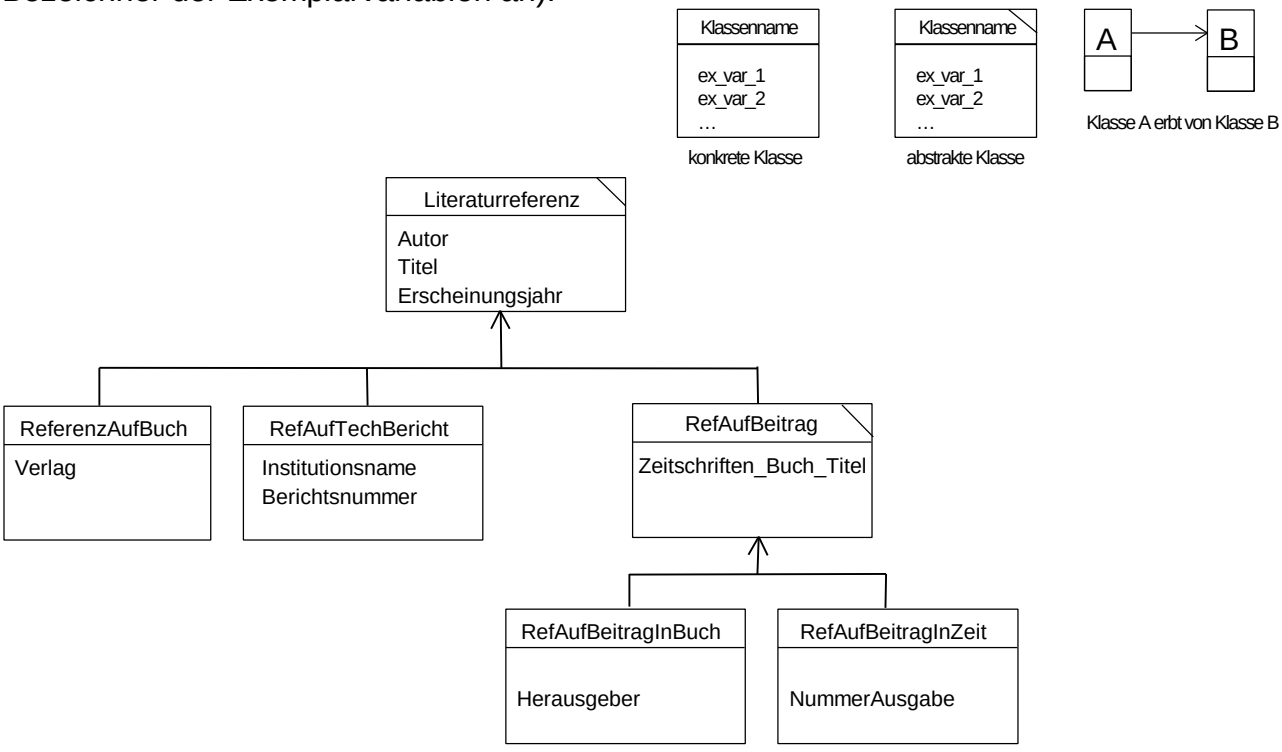
Referenz auf einen Technischen Bericht: diese ist charakterisiert durch Angabe von Autor, Titel, Erscheinungsjahr, Name der Institution und Nummer des Berichts.

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.

7.1(5 Pkt.) Entwerfen Sie eine geeignete Klassenhierarchie, um die oben aufgelisteten Arten an Literaturreferenzen zu realisieren.

Achten Sie dabei darauf, daß gemeinsame Merkmale in abstrakten Klassen zusammengezogen 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).



Vorname	Name	Fach	Sem.	Matr.-Nr

7.2 (4 Pkt.) 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 `initialsiere` (initialsiert 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;
```

Vorname	Name	Fach	Sem.	Matr.-Nr

7.3 (5 Pkt.) Implementieren Sie die zur Nachricht `alsText` gehörende Prozedur der Klasse `ReferenzAufBeitragInZeitschrift.`

```
PROCEDURE AlsText(self: ReferenzAufBeitragInZeitschrift): TEXT=
BEGIN
  RETURN ReferenzAufBeitrag.alsText() & ", " &
         self.gibNummerAusgabe();
END AlsText;
```

Vorname	Name	Matr.-Nr

Hinweise zur Bearbeitung:

- Schreiben Sie auf jedes Blatt Vorname, Name und Matrikelnummer.
- Beantworten Sie die Fragen lesbar und auch **im Detail** korrekt.
- Bitte beantworten Sie die Aufgaben auf den **Aufgabenblättern** (benutzen Sie auch die Rückseiten). Antworten auf anderen Blättern können nur berücksichtigt werden, wenn **Name, Matrikelnummer und Aufgabennummer** deutlich darauf erkennbar ist.
- Was nicht bewertet werden soll, kennzeichnen Sie bitte durch **Durchstreichen**.
- Werden Täuschungsversuche beobachtet, so wird die Klausur mit nicht bestanden bewertet.
- Kennzeichnen Sie alle Blätter mit **Namen** und **Matrikelnummer**.
- Geben Sie am Ende der Klausur alle Blätter zusammen mit dem Deckblatt ab.

Die Organisatoren wünschen allen Kandidaten viel Erfolg!

	Anzahl Punkte	Erreichte Punkte
Aufgabe 1	**8**	
Aufgabe 2	**10**	
Aufgabe 3	**10**	
Aufgabe 4	**6**	
Aufgabe 5	**12**	
Aufgabe 6	**10**	
Aufgabe 7	**14**	
SUMME	**70**	

Vorname	Name	Matr.-Nr

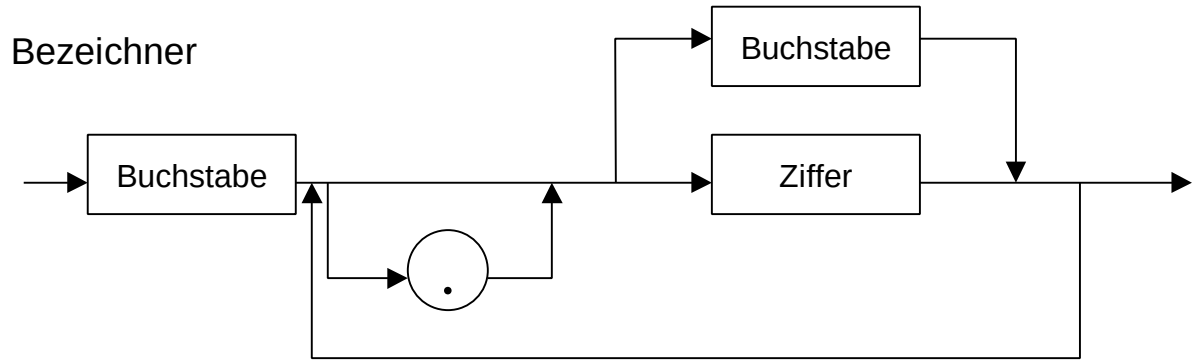
Aufgabe 1 : Syntaxdiagramm und EBNF

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 (4 Pkt.) Geben Sie die Syntax für Bezeichner durch ein Syntaxdiagramm an! Die Lösung muß knapp und eindeutig sein.



1.2 (4 Pkt.) 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 }]

Vorname	Name	Matr.-Nr

Aufgabe 2: Eine einfache Funktion

Die Funktion J mit einer CARDINAL-Zahl als Argument sei wie folgt definiert:

- J(X) = 0, wenn x eine Quadratzahl ist (z.B. J(1) = J(4) = J(9) = 0)
- Andernfalls ist J(x) um 1 größer als J(d), wenn d die Differenz zwischen x und der nächstgrößeren Quadratzahl ist. Z.B. ist J(12) = 1, J(13) = 2, J(101) = 3

2.1 (10 Pkt.) Implementieren Sie eine Modula-3 Funktion zur Berechnung von J. Verwenden Sie nur INTEGER bzw CARDINAL-Arithmetik.

```
PROCEDURE J (x : CARDINAL) : CARDINAL =
VAR I, resultat : CARDINAL;
BEGIN
  i := 1;
  WHILE (x > i * i) DO
    i := i + 1;
  END;                    (* i*i ist entweder gleich x oder *)
  IF (x = i * i)THEN      (* naechstgroessere Quadratzahl   *)
    resultat := 0;
  ELSE
    resultat := J((i * i) - x) + 1;
  END;
  RETURN resultat;
END J_MS;
```

Vorname	Name	Matr.-Nr

Aufgabe 3: Erkennen eines Palindroms

Ein Palindrom ist eine Folge von Buchstaben, die vorwärts und rückwärts gelesen den gleichen Sinn ergibt.
Beispiele für Palindrome sind: Anna, Reliefpfeiler, AnnaHetzteHanna,

3.1 (10 Pkt.) Schreiben Sie eine Modula.3-Funktion `Palindrom`, die erkennt, ob die Buchstaben eines Arrays ein Palindrom bilden oder nicht.

Folgende Deklarationen stehen dazu zur Verfügung:

```
CONST Anzahl = 39;
TYPE Zeichenfeld : ARRAY [0 .. Anzahl] OF CHAR;
```

Weiterhin gelte folgendes:

- In einem Zeichenfeld sind nur Großbuchstaben enthalten (mindestens einer).
- Zwischen den Großbuchstaben stehen keine Leerzeichen.
- Ist ein Zeichenfeld nicht vollständig besetzt, so ist der Rest mit Leerzeichen aufgefüllt.

```
PROCEDURE Palindrom (text: Zeichenfeld) : BOOLEAN =
VAR laenge, i : INTEGER := 0;
    gleich : BOOLEAN := TRUE;
BEGIN
  (*Laenge bestimmen *)
  WHILE (laenge <= Anzahl) AND (text[laenge] # ' ') DO
    laenge := laenge + 1;
  END;

 (* pruefen, ob von vorne und von hinten die Zeichen gleich sind*)
  WHILE (i <= laenge - 1) AND gleich DO
    IF (text[i] # text[laenge -1 -i]) THEN
      gleich := FALSE;
    END;
    i := i + 1;
  END;

  RETURN gleich;
END Palindrom;
```

Vorname	Name	Matr.-Nr

Aufgabe 4: Matrixtransformation

Gegeben sei eine (m, n)-Matrix A (die Normalmatrix). Dazu kann die transponierte (n, m)-Matrix A^T berechnet werden.
Beispiel:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix} \qquad A^T = \begin{pmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{pmatrix}$$

4.1 (2 Pkt.) Geben Sie alle notwendigen Deklarationen für den Typ `Normalmatrix` und den Typ `Transponiertematrix` an!

```
CONST M = 2;
      N = 3;
TYPE MDim = [1 .. M];
     NDim = [1 .. N];

TYPE Normalmatrix = ARRAY MDim, NDim OF INTEGER;
     Transponiertematrix = ARRAY NDim, MDim OF INTEGER;
```

4.2 (4 Pkt.) Implementieren Sie die Modula-3 Funktion `Transponiere` auf der Basis der von Ihnen gewählten Deklarationen!

```
PROCEDURE Transponiere (nm : Normalmatrix): Transponiertematrix =
  VAR tm : Transponiertematrix;
BEGIN
  FOR i := FIRST(MDim) TO LAST(MDim) DO
    FOR j := FIRST(NDim) TO LAST(NDim) DO
      tm[j, i] := nm[i, j];
    END;
  END;
  RETURN tm;
END Transponiere;
```

Vorname	Name	Matr.-Nr

Aufgabe 5: Schleifen

Gegeben ist ein Feld F [1 .. 100] aus INTEGER-Werten.

Eine Funktion `Produkt` soll das Produkt der Zahlen des Feldes bis zur ersten Null liefern. Beispiel: Steht die erste Null an der Stelle 20, so liefert die Funktion das Produkt der Werte F[1] bis F[19]. Enthält das Feld keine Null, so liefert die Funktion das Produkt aller Elemente. Kann kein Produkt berechnet werden (z.B. Null steht an erster Stelle), dann liefere die Funktion den Wert 1.

5.1 (12 Pkt.) Realisieren Sie für alle Schleifenarten, die Sie kennen, je eine Funktion `Produkt`!

```
TYPE F = ARRAY [1 .. 100] OF INTEGER;

PROCEDURE ForProdukt(zf : F) : INTEGER =
VAR produkt : INTEGER := 1;
    vorErsterNull : BOOLEAN := TRUE;
BEGIN
  FOR i := 1 TO 100 DO
    IF zf[i] # 0 THEN
      IF vorErsterNull THEN
        produkt := produkt * zf[i];
      END;
    ELSE
      vorErsterNull := FALSE;
    END;
  END;
  RETURN produkt;
END ForProdukt;

PROCEDURE WhileProdukt(zf : F) : INTEGER =
VAR produkt : INTEGER := 1;
    i : INTEGER := 1;
BEGIN
  WHILE (i <= 100) AND (zf[i] # 0) DO
    produkt := produkt * zf[i];
    i := i + 1;
  END;
  RETURN produkt;
END WhileProdukt;
```

Vorname	Name	Matr.-Nr

```
PROCEDURE RepeatProdukt(zf : F) : INTEGER =
VAR produkt : INTEGER := 1;
    i : INTEGER := 1;
BEGIN
  IF zf[i]# 0 THEN
    REPEAT
      produkt := produkt * zf[i];
      i := i + 1;
    UNTIL (i > 100) OR (zf[i] = 0);
  END;
  RETURN produkt;
END RepeatProdukt;

PROCEDURE LoopProdukt(zf : F) : INTEGER =
VAR produkt : INTEGER := 1;
    i : INTEGER := 1;
BEGIN
  LOOP
    IF zf[i] # 0 THEN
      produkt := produkt * zf[i];
    END;
    IF (i = 100) OR (zf[i] = 0) THEN EXIT; END;
    i := i + 1;
  END;
  RETURN produkt;
END LoopProdukt;
```

Vorname	Name	Matr.-Nr

Aufgabe 6: Listen

Ein Eisenbahnzug sei durch eine einfach verkettete Liste dargestellt. Dazu stehen die folgenden Deklarationen zur Verfügung:

```
TYPE LokKennung = TEXT;
     Wagennummer = [1 .. 999];
     WagenRef = REF Wagen;
     Wagen = RECORD
               wagennr : Wagennummer;
               naechsterWagen : WagenRef;
             END
     Zug = RECORD
             lokName : LokKennung;
             wagen : WagenRef;
           END;
```

Die Wagennummern sind frei gewählt, aber eindeutig. Ein möglicher Zug kann somit aus folgenden Elementen bestehen: Lokomotive „ICE 210“, gefolgt von den Wagen mit den Nummern 200, 231, 100, 55, 260.

6.1 (4 Pkt.) Schreiben Sie eine Prozedur, die die tatsächlichen Bestandteile (Lokomotive und Wagen) eines Zuges auf dem Bildschirm ausgibt.

```
PROCEDURE Ausgeben (zug : Zug)=
VAR aktWagen : WagenRef;
BEGIN
  SIO.Putline (zug.lokName);
  aktWagen := zug^.wagen;
  WHILE aktWagen # NIL DO
    SIO.Putline (aktWagen^.wagennr);
    aktWagen := aktWagen^.naechsterWagen;
  END;
END Ausgeben;
```

Vorname	Name	Matr.-Nr

6.2 (6 Pkt.) Schreiben Sie eine Prozedur `Umkehren`, die zwei Züge als Parameter hat und folgende Anforderungen erfüllt:

a) der zweite Zug hat vor dem Prozeduraufruf keine Wagen
b) der erste Zug hat nach dem Prozeduraufruf keine Wagen
c) der zweite Zug hat nach dem Prozedurauf die Wagen des ersten Zuges in umgekehrter Reihenfolge (Gemäß obigem Beispiel entsteht an der zweiten Lokomotive die Wagenfolge 260, 50,100, 230, 200)

```
PROCEDURE Umkehren (VAR zug1, zug2 : Zug) =
VAR wagen : WagenRef;
BEGIN
  WHILE zug1.wagen # NIL DO
    wagen := zug1.wagen;
    zug1.wagen := wagen^.naechsterWagen;
    wagen^.naechsterWagen := zug2.wagen;
    zug2.wagen := wagen;
  END;
END Umkehren;
```

Vorname	Name	Matr.-Nr

Aufgabe 7: 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.

7.1(5 Pkt.) 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.
```

7.2 (4 Pkt.) 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 Element = BRANDED REF RECORD
                   t : TEXT;
                   naechstesElement : SetOfText;
                 END;
```

Vorname	Name	Matr.-Nr

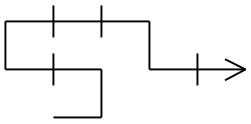
7.3 (5 Pkt.) Implementieren Sie die Funktion der Mengenvereinigung des ADTs `SetOfText` gemäß Ihrer Deklaration im Interface-Modul!

```
PROCEDURE Vereinigung (s1, s2 : SetOfText) : SetOfText =
VAR resSet : SetOfText;
BEGIN
  resSet := Erzeuge();
  WHILE s1 # NIL DO
    Aufnehmen(resSet, s1.t);
    s1 := s1^.naechstesElement;
  END;
  WHILE s2 # NIL DO
    Aufnehmen(resSet, s2.t);
    s2 := s2^.naechstesElement;
  END;
  RETURN resSet;
END Vereinigung;
```

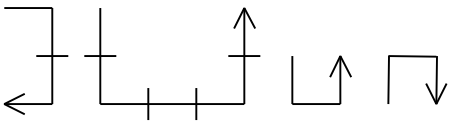
Vorname	Name	Matr.-Nr	1

Aufgabe 1: Syntax

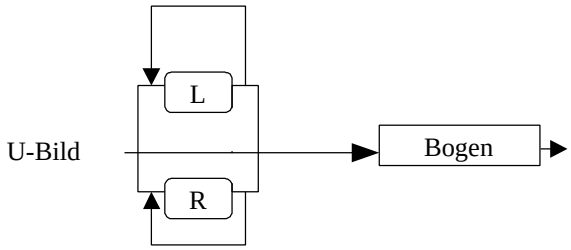
Ein Zeichenkopf werde durch die Anweisungen Links (L), Rechts(R) und Schritt (S) gesteuert. Schritt bewirkt einen Schritt in der bisherigen Richtung, Links und Rechts bewirken eine Drehung um 90 Grad. Alle Schritte sind gleich lang, die Länge eines Schrittes sei a. Zu Beginn ist die Richtung horizontal nach rechts. Beispielsweise entsteht durch die Anweisungsfolge SLSLSSRSRSSSRSLSS die folgende Graphik.

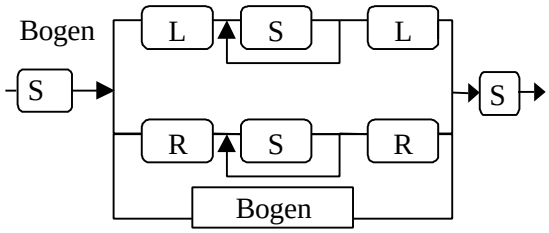


Mit diesen Mitteln kann man auch U-förmige Gebilde erzeugen (siehe nachfolgende Figuren).



1.1 (4 Pkt) Geben Sie eine Syntaxdefinition (mit Hilfe von Syntaxdiagrammen) an, mit der alle möglichen U-Bilder (in beliebiger Größe und Drehung) erzeugt werden können.





Vorname	Name	Matr.-Nr	2

1.2 (5 Pkt) Nachfolgend sind einige Formen vorgegeben. Geben Sie bitte für die Formen (a-c) die entsprechende Syntaxbeschreibung in EBNF an!
Hinweis: Diese Formen sollen aus der Ausgangsstellung (also horizontal nach rechts) erzeugt werden; Drehungen müssen nicht berücksichtigt werden.

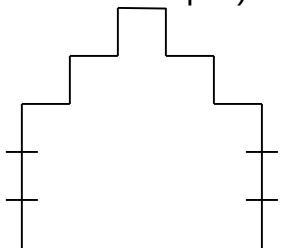
a) Alle Rechtecke mit der Höhe 3*a, Breite n*a, n > 0.

Rechteck = "L" "S" "S" "S" "L" Seite.
Seite = ("S" Seite "S" | "S" "L" "S" "S" "S" "L" "S").

b) Alle Treppen mit konstanter Breite und Höhe der Stufen (jeweils 2*a), 0 bis beliebig viele Stufen.

Treppe = { "L" "S" "S" "R" "S" "S" }.

c) Alle Türme mit einem geraden beliebig langen Schaft, dann einer symmetrischen Treppe aus Einzelstufen, an der Spitze mit der Breite a. (siehe abgebildetes Beispiel)



Turm = "L" Schaft.
Schaft = ("S" Schaft "S" | SymTreppe "R").
SymTreppe = ("R" "S" "L" "S" SymTreppe "R" "S" "L" "S" | Spitze).
Spitze = "R" "S"

Vorname	Name	Matr.-Nr	3

Aufgabe 2: Rekursion

In der Familie Albertoni, in der seit Urzeiten nur Frauen etwas gelten, bekommt jedes weibliche Familienmitglied im Laufe ihres Lebens zwei Töchter, und zwar stets die erste Tochter im Alter von 20 und die zweite im Alter von 23 Jahren.

2.1 (3 Pkt) Schreiben Sie eine Modula-3–Funktion zur Berechnung der Anzahl aller weiblichen Familienmitglieder (die jemals gelebt haben ...), wenn das Geburtsjahr der Urmutter Eva Albertoni gegeben ist. Das Jahr, in dem wir uns gerade befinden, ist selbstverständlich auch gegeben.

```
CONST AktJahr = 1999;

PROCEDURE AnzahlFrauen (gebjahr : INTEGER): INTEGER =
VAR diff : INTEGER := AktJahr - gebjahr;
BEGIN
  IF diff >= 23 THEN
    RETURN AnzahlFrauen(gebjahr+20) +
           AnzahlFrauen(gebjahr+23) + 1;
  ELSIF diff >= 20 THEN
    RETURN AnzahlFrauen(gebjahr+20) + 1;
  ELSE
    RETURN 1;
  END;
END AnzahlFrauen;
```

Vorname	Name	Matr.-Nr	4

Gegeben sei ein Schachbrett der Größe acht mal acht, die Felder des Bretts seien mit (1*;* 1) bis (8*;* 8) bezeichnet. Auf diesem Schachbrett tummle sich nun ein einzelner Springer. Die Frage ist, welche Felder der Springer bei gegebener Startposition in *n* Zügen erreichen kann.

2.2 (5 Pkt) Schreiben Sie eine Modula-3-Prozedur `Springer`, die als Eingabe die Koordinaten eines Startfeldes (`XStart ;YStart`) sowie eine natürliche Zahl *n* erhält. Diese soll die Koordinaten aller Felder ausgeben, die ein Springer innerhalb von (höchstens) *n* Zügen vom Startfeld aus erreichen kann. Doppelnennungen sind erlaubt.

```
PROCEDURE PositionGueltig (xStart, yStart : INTEGER): BOOLEAN =
BEGIN
  RETURN (xStart > 0) AND (xStart < 9) AND
         (yStart > 0) AND (yStart < 9);
END PositionGueltig;


PROCEDURE GebePositionAus(xStart, yStart: INTEGER) =
BEGIN
  SIO.PutText("x= "); SIO.PutInt(xStart); SIO.PutText(" ");
  SIO.PutText("y= "); SIO.PutInt(yStart); SIO.Nl();
END GebePositionAus;


PROCEDURE Springer (xStart, yStart : INTEGER; n : INTEGER) =
BEGIN
  IF n >= 0 THEN
    IF PositionGueltig (xStart, yStart)THEN
      GebePositionAus(xStart, yStart);
      FOR i := -2 TO 2 DO
        IF (i #0) THEN
          Springer (xStart+i, yStart+3-ABS(i), n-1);
          Springer (xStart+i, yStart-3+ABS(i), n-1);
        END;
      END;
    END;
  END;
END Springer;
```

Vorname	Name	Matr.-Nr	5

Aufgabe 3: Codenummern

Als Codenummern für Kreditkarten möchte eine Bank Zahlen mit bestimmten Quersummen einsetzen. Die Quersumme ist als Summe der Ziffern definiert. Beispiel: Die Quersumme von 4711 ist 4+7+1+1 = 13. Dummerweise vergaßen die Ingenieure bei der Planung der Bankomaten die Taste mit der "0". Deshalb ist die Bank nur an Zahlen ohne die Ziffer 0 interessiert.

Beispiel: Quersumme = 4
Zahlen: 1111, 112, 121, 13, 211, 2 2, 31, 4

3.1 (8 Pkt) Schreiben Sie eine Modula-3 Prozedur, welche für eine ganzzahlige Quersumme (maximal 100) alle Zahlen mit dieser Quersumme, welche die Ziffer 0 nicht enthalten, ausgibt. Die Zahlen sollen wie im Beispiel in lexikographischer Reihenfolge ausgegeben werden.

```
TYPE Codezahl = ARRAY [1 .. 100] OF CHAR ;


PROCEDURE AlsChar(zahl: INTEGER): CHAR =
BEGIN
  RETURN VAL(zahl+ORD('0'), CHAR);
END AlsChar;


PROCEDURE Ausgeben(VAR aus: Codezahl; pos : INTEGER) =
BEGIN
  FOR i := 1 TO pos DO
    SIO.PutChar(aus[i]);
  END;
  SIO.Nl();
END Ausgeben;


PROCEDURE Codenummern (quer : INTEGER; VAR ausgabe : Codezahl;
                       pos  : INTEGER) =
BEGIN
  FOR i := 1 TO MIN(quer, 9) DO
    IF (quer - i) > 0 THEN
      ausgabe[pos + 1] := AlsChar(i);
      Codenummern (quer - i, ausgabe, pos + 1);
    ELSIF (quer - i) = 0 THEN
      ausgabe[pos + 1] := AlsChar(i);
      Ausgeben (ausgabe, pos+1);
    END;
  END;
END Codenummern;
```

Vorname	Name	Matr.-Nr	6

Aufgabe 4: Hausbau

Bei einem Hausbau strebt die Bauleitung eine möglichst rasche Fertigstellung an. Jeder beteiligte Handwerker gibt an, wie lange seine Arbeit dauert und welche anderen Arbeiten beendet sein müssen, bevor er mit der Arbeit beginnen kann. Diese Information wird in einer Tabelle abgelegt, die beispielsweise folgendermaßen aussehen kann:

Handwerker	Dauer	Beendet sein müssen
Maler	2 Wochen	Gipser, Schreiner, Fliesenleger
Dachdecker	2 Wochen	Maurer
Elektriker	4 Wochen	Maurer
Gipser	3 Wochen	Elektriker, Maurer
Kücheneinrichter	1 Woche	Maler
Maurer	16 Wochen	-
Schreiner	3 Wochen	Maurer, Gipser, Fliesenleger, Elektriker
Fliesenleger	4 Wochen	Maurer, Gipser

Dabei ist sichergestellt, daß es keine Situation gibt, die dazu führt, daß die Arbeit nicht fortgesetzt werden kann (z.B. zwei Handwerker warten gegenseitig aufeinander). Die Bauleitung will für eine beliebige Arbeitstabelle wissen, wie lange der Hausbau bei bestmöglicher Koordination der Arbeiten dauert.

4.1 (3 Pkt) Entwerfen Sie in Modula-3 Datenstrukturen, die geeignet sind, diese Informationen aufzunehmen. Kommentieren Sie diese!

```
TYPE Handwerker = { Maler, Dachdecker, Elektriker, Gipser,
                    Kuecheneinr, Maurer, Schreiner, Fliesenleger};

     HWMenge = SET OF Handwerker;

     (* Dauer der Aktivität eines Handwerkers und Menge der
        Arbeiten, die vorher beendet sein müssen *)
     Aktivitaet = RECORD
                    dauer : INTEGER;
                    beendetSeinMuss := HWMenge{};
                  END;

     (* Tabelle der Aktivitäten aller am Bau beteiligten
        Handwerker *)
     Arbeitstabelle = ARRAY [FIRST(Handwerker)..LAST(Handwerker)]
                        OF Aktivitaet;
```

Vorname	Name	Matr.-Nr	7

4.2 (7 Pkt) Realisieren Sie eine Modula-3 Funktion, die die von Ihnen deklarierten Datenstrukturen verwendet und die minimale Bauzeit ermittelt.

```
VAR bereitsBeendet := HWMenge{}; (* Zu Beginn die leere Menge *)

    nochNichtBeendet := HWMenge {Handwerker.Maler,
      Handwerker.Dachdecker, Handwerker.Elektriker,
      Handwerker.Gipser, Handwerker.Kuecheneinr,
      Handwerker.Maurer, Handwerker.Schreiner,
      Handwerker.Fliesenleger};  (* Zu Beginn alle Handwerker *)


PROCEDURE Bauzeit (arbtab : Arbeitstabelle): INTEGER =

VAR baldBeendet : HWMenge; (* Menge der Handwerker, die im
                              nächsten Schritt anfangen können *)
    max, minDauer : INTEGER := 0;

BEGIN
  WHILE (nochNichtBeendet # HWMenge{})DO
    max := 0;
    baldBeendet := HWMenge{};
    FOR hw := FIRST(Handwerker) TO LAST(Handwerker) DO
      IF hw IN nochNichtBeendet THEN
        (* Ist beendetSeinMuss Teilmenge von bereitsBeendet? *)
        IF arbtab[hw].beendetSeinMuss <= bereitsBeendet THEN
          (* Ausgeben der Aktivität *)
          SIO.PutText(Drucktabelle[hw]);
          (* Prüfe, ob die neue Aktivität am längsten dauert *)
          IF arbtab[hw].dauer > max THEN
            max := arbtab[hw].dauer;
          END;
          (* Füge HW zu den jetzt beginnenden Handwerkern hinzu *)
          baldBeendet := baldBeendet + HWMenge{hw};
        END;
      END;
    END;
    (* Korrigiere Mengen bereitsBeendet und nochNichtBeendet *)
    bereitsBeendet := bereitsBeendet + baldBeendet;
    nochNichtBeendet := nochNichtBeendet - baldBeendet;
    (* Addiere maximale Aktivitätendauer zur minimalen Bauzeit *)
    minDauer := minDauer + max; SIO.Nl();
  END;

  RETURN minDauer;
END Bauzeit;
```

Vorname	Name	Matr.-Nr	8

Aufgabe 5: Immobilienverwaltung

Die Firma BonnMobilia möchte ihre Kunden- und Immobilienverwaltung automatisieren. Dazu soll ein Programm erstellt werden, daß es erlaubt, beliebig viele Kunden und beliebig viele Immobilien zu verwalten. Weiterhin soll dieses Programm die folgenden Anforderungen erfüllen:

a) Jeder Kunde hat einen Namen und eine Adresse und kann beliebig viele Immobilien besitzen.

b) Jede Immobilie hat eine Nummer, eine Adresse, einen Wert und kann beliebig viele Eigentümer haben.

Immobilien- und Kundeninformationen sollen so miteinander verknüpft sein, daß Änderungen an bestehenden Immobilien (z.B. ihr Wert) und Änderungen an Kundeninformationen (z.B. neue Adresse) nur an einer Stelle vorgenommen werden müssen. Wenn also beispielsweise die Kunden A und B die Immobilie C besitzen und deren Wert geändert wird, dann muß diese Veränderung auch aus den Informationen zu den Kunden A und B ersichtlich sein.

5.1 (5 Pkt) Entwerfen Sie in Modula-3 Datentypen, die geeignet sind, die Immobilien- und Kundeninformationen zu verwalten und die oben genannten Anforderungen berücksichtigen. Kommentieren Sie die einzelnen Datentypen!

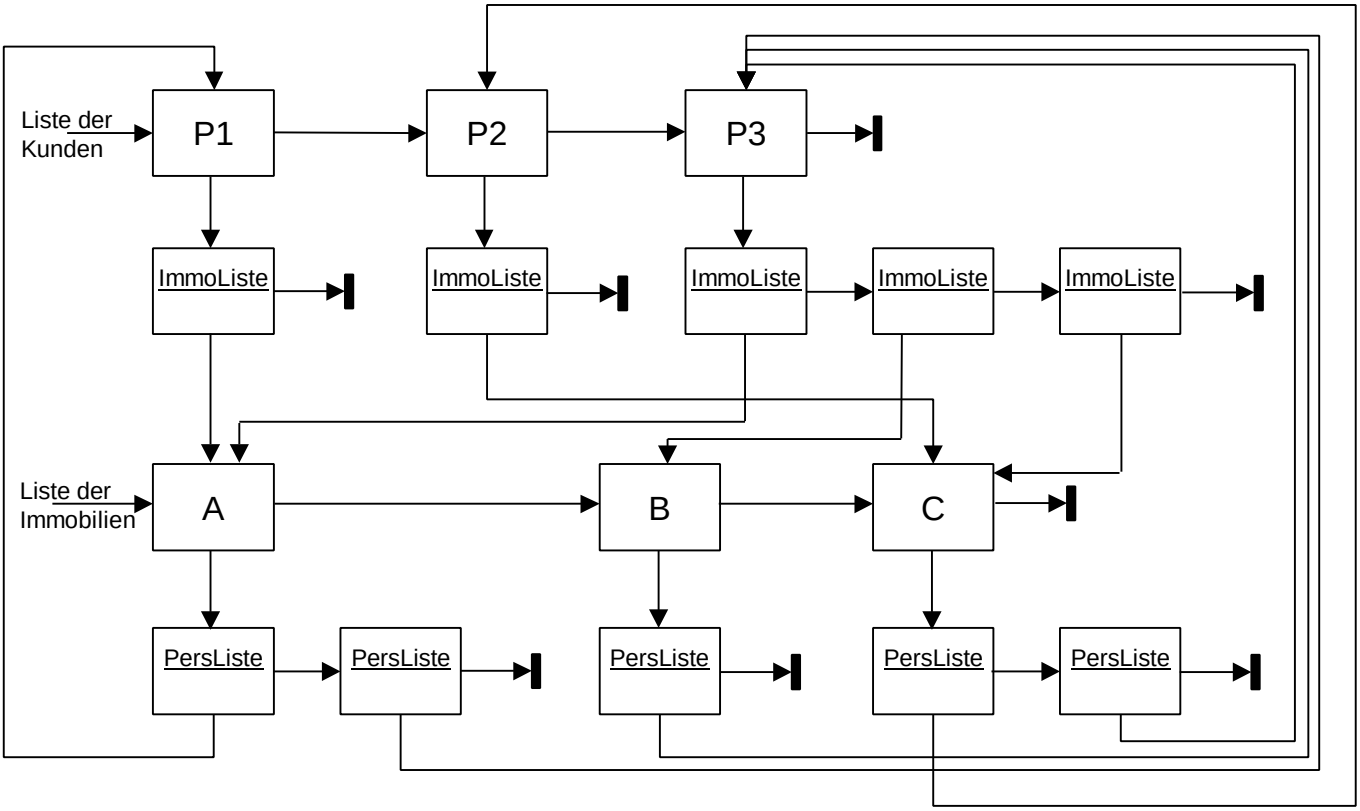
```
TYPE

Person =
  RECORD
    name, adresse : TEXT;
    immobilien : ImmoListe;
    nachfolger : PersRef;
  END;

PersRef = REF Person;

(* Element Besitzerliste *)

PersListe = REF
  RECORD
    persRef    : PersRef;
    nachfolger : PersListe;
  END;
```

```
Immobilie =
  RECORD
    nr : INTEGER;
    adresse : TEXT;
    wert : REAL;
    besitzer : PersListe;
    nachfolger: ImmoRef;
  END;

ImmoRef = REF Immobilie;

(* Element der Besitzliste *)

ImmoListe = REF
  RECORD
    immoRef    : ImmoRef;
    nachfolger : ImmoListe;
  END;
```

Vorname	Name	Matr.-Nr	9

Das neue Programm verwalte drei Kunden und vier Immobilien in der folgenden Konstellation: Kunde P1 besitzt die Immobilie A, Kunde P2 besitzt die Immobilie C, Kunde P3 besitzt die Immobilien A, B und C.

5.2 (2 Pkt) Stellen Sie die oben beschriebene Situation auf der Basis der von Ihnen gewählten Datenstrukturen grafisch dar. Verwenden Sie dazu die aus der Vorlesung bekannte "Kästchen-Pfeil"-Notation.



Vorname	Name	Matr.-Nr	10

5.3 (5 Pkt) Schreiben Sie eine Modula-3 Prozedur `ListeAlleImmobilien`, die als Parameter die Liste aller verwalteten Immobilien erhält (diesen Datentyp haben Sie deklariert), und für alle Immobilien alle ihre Besitzer im folgenden Format ausgibt:
<Nr der Immobilie>
Besitzer 1 : <Name des Besitzers>
...
Besitzer n : <Name des Besitzers>

```
PROCEDURE ListeAlleImmobilien (immobilien : ImmoRef) =

VAR immo     : ImmoRef;
    i        : INTEGER;
    besitzer : PersListe;

BEGIN
  immo := immobilien;

  WHILE immo # NIL DO
    SIO.PutInt (immo^.nr); SIO.Nl();
    i := 1;
    besitzer := immo^.besitzer;

    WHILE besitzer # NIL DO
      SIO.PutText("Besitzer "); SIO.PutInt(i);
      SIO.PutText(": " & besitzer^.persRef^.name); SIO.Nl();
      besitzer := besitzer^.nachfolger;
      i := i + 1;
    END;

    immo := immo^.nachfolger;
  END;

END ListeAlleImmobilien;
```

Vorname	Name	Matr.-Nr	11

Aufgabe 6: Einfach verkettete 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 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
$z = (x_1,y_1, x_2,y_2,\ldots,x_n,y_n,y_{n+1}, \ldots, y_m)$	falls m > n
$z = (x_1,y_1, x_2,y_2,\ldots,x_m,y_n)$	falls n = m

6.1 (4 Pkt) 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 Eingabelisten werden dabei zerstört)

```
TYPE Liste         = REF ListenElement;

     ListenElement = RECORD
                       inhalt : INTEGER;
                       nachfolger : Liste;
                     END;

PROCEDURE ErzeugeZ( VAR x,y,z : Liste)=
VAR hz : Liste;

BEGIN
  z := x;
  x := x^.nachfolger;
  z^.nachfolger := y;
  y := y^.nachfolger;
  hz := z^.nachfolger;
  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;

  IF x = NIL THEN
    hz^.nachfolger := y;
  ELSE
    hz^.nachfolger := x;
  END;

END ErzeugeZ;
```

Lösungen zu Übung 1

Lösung 1.1: Zahlensysteme

Sind die drei Zahlen in den Zahlensystemen mit den Basen a, b und c, so muß gelten:

$2 * a + 1 = 1 * b + 0 = 2 * c + 3$ wobei: $a > 2,\ b > 1,\ c > 3$

Die Lösung wird dominiert durch die Randbedingung c > 3, also:

c = 4, woraus b = 11 und a = 5 resultieren.

Lösung 1.2: Binärbrüche

Berücksichtigt man jeweils die positionsabhängige Gewichtung und das Vorzeichen, ergibt sich:

-1.0000 $\Rightarrow$ $(-1) * (1 * 2^0) = (-1) * 1 = \underline{-1}$

0.1 $\Rightarrow$ $(+1) * (0 * 2^0 + 1 * 2^{-1}) = 1/2 = \underline{0.5}$

0.0001 $\Rightarrow$ $(+1) * (0 * 2^0 + 0 * 2^{-1} + 0 * 2^{-2} + 0 * 2^{-3} + 1 * 2^{-4}) = 1/16 = \underline{0.0625}$

-1.001 $\Rightarrow$ $(-1) * (1 * 2^0 + 0 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}) =$
$(-1) * (1 + 1/8) = -9/8 = \underline{-1.125}$

-0.101 $\Rightarrow$ $(-1) * (0 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3}) =$
$(-1) * (1/2 + 1/8) = -5/8 = \underline{-0.625}$

1.1011 $\Rightarrow$ $(+1) * (1 * 2^0 + 1 * 2^{-1} + 0 * 2^{-2} + 1 * 2^{-3} + 1 * 2^{-4}) =$
$(1 + 1/2 + 1/8 + 1/16) = 1\ 11/16 = \underline{1.6875}$

-0.1101 $\Rightarrow$ $(-1) * (0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} + 0 * 2^{-3} + 1 * 2^{-4}) =$
$(-1) * (1/2 + 1/4 + 1/16) = -13/16 = \underline{-0.8125}$

Die letzte Zahl ist ein periodischer Binärbruch und stellt eine Annäherung an den Dezimalwert 0.1 dar:

0.000110011… $\Rightarrow$

$(+1) * (0 * 2^0 + 0 * 2^{-1} + 0 * 2^{-2} + 0 * 2^{-3} + 1 * 2^{-4} + 1 * 2^{-5} + 0 * 2^{-6} + 0 * 2^{-7}$
$+ 1 * 2^{-8} + 1 * 2^{-9} + \ldots) \rightarrow \underline{0.1}$

Die Vermutung läßt sich leicht nachprüfen, indem man versucht 0.1 in eine Binärzahl umzurechnen. Dabei gilt, multipliziere die Nachkommastellen mit der neuen Basis g=2, notiere und streiche jeweils den entstehenden Überlauf links des Kommas bis keine von null verschiedenen Nachkommastellen mehr existieren. Die notierten Überläufe ergeben von links nach rechts gelesen die Nachkommastellen des Binärbruchs:

0.1 * 2 =	0.2	$\rightarrow$	0
0.2 * 2 =	0.4	$\rightarrow$	0
0.4 * 2 =	0.8	$\rightarrow$	0
0.8 * 2 =	1.6	$\rightarrow$	1
0.6 * 2 =	1.2	$\rightarrow$	1
0.2 * 2 =	0.4	$\rightarrow$	0

…

Lösung 1.3: Algorithmus

```
WENN Automat in Betrieb ist DANN
  führe Karte in Automat ein,
  WENN Karte lesbar DANN
    WIEDERHOLE
      eingeben der PIN
    BIS PIN korrekt ODER drei Eingaben erfolgt sind,
    WENN PIN korrekt DANN
      Betrag eingeben,
      WENN Betrag nicht genehmigt DANN
        anderen Betrag eingeben,
      WENN Betrag genehmigt DANN
        WENN Karte ausgegeben DANN
          Karte entnehmen,
          WENN Geld ausgegeben DANN
            Geld entnehmen und nachzählen,
            WENN Geldbetrag korrekt DANN
              alles klar!
            SONST reklamieren
          SONST reklamieren
        SONST reklamieren
      SONST
        nach Abbruch ausgegebene Karte entnehmen
    SONST
      neue Karte beantragen,
  SONST
    WENN Karte falsch rum DANN
      Karte drehen und erneut versuchen
    SONST
      reklamieren
SONST
  anderen Automat suchen.
```

Lösung 1.4: Algorithmus

```
Suche zum Begriffsanfang passendes Lexikon im Regal,
WENN passendes Lexikon gefunden DANN
  nehme Lexikon aus dem Regal,
  schlage Lexikon etwa bei der halben Seitenzahl auf,
  WIEDERHOLE
    WENN Begriff in lexikal. Ordnung weiter vorne DANN
      schlage Lexikon um etwa halbierte Seitenanzahl vom
      letzten Versuch weiter vorne auf
    SONST
      schlage Lexikon um etwa halbierte Seitenanzahl vom
      letzten Versuch weiter hinten auf
  BIS Seite die Begriff enthalten muß gefunden,
  betrachte ersten Begriff auf der Seite,
  SOLANGE der betrachtete Begriff nicht der gesuchte
          Begriff ist
      UND noch nicht alle Begriffe auf der Seite
          betrachtet wurden,
    betrachte den darauffolgenden Begriff auf der Seite,
  WENN der zuletzt betrachtete Begriffe der gesuchte
       Begriff ist DANN
    WENN mehrere Definitionen gegeben sind DANN
      betrachte die erste Definition,
      SOLANGE die betrachtete Definition nicht die
              gesuchte Definition ist
          UND noch nicht alle Definitionen betrachtet
              wurden,
        betrachte die darauffolgende Definition,
      WENN die zuletzt betrachtete Definition die gesuchte
           Definition ist DANN
        alles klar!
      SONST
        reklamiere beim Verlag
  SONST
    reklamiere beim Verlag
SONST
  reklamiere bei Bibliotheksleitung.
```

Lösungen zu Übung 2

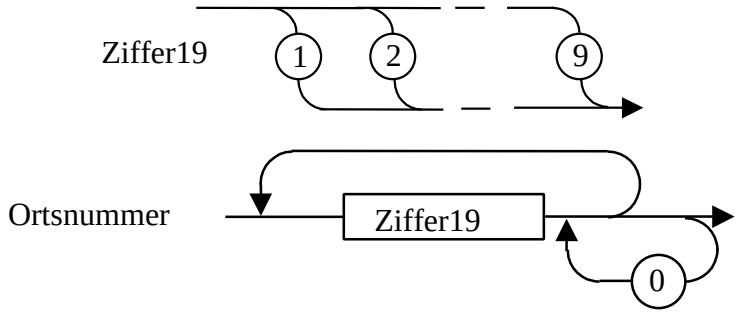
Lösung 2.1: Sprache

Unterstrichen dargestellte Zeichen müssen gestrichen werden, damit der Satz korrekt wird:

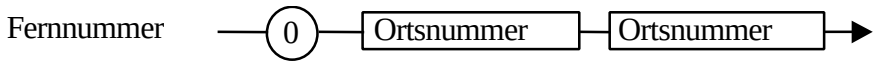
a) falsch A L Z
b) richtig A (M) Z
c) falsch (A N (A N (K) Z) Z)
d) falsch A (N N N N (A (L) Z) Z
e) falsch A N N K (A N N (A (N (K)) Z) Z) Z
f) falsch Z (A N (N) Z) A ⇒ Keine Korrektur möglich!

Lösung 2.2: Syntax von Telefonnummern

a) Ortsgespräch



b) Ferngespräch



c) Auslandsgespräch

Lösung 2.3: Grammatik

Das Alphabet der Sprache S ist auch die Menge der Terminalsymbole:

T = { a, b, c, (,), [,] }

Variante 1: G = {N, T, P, A} mit
N = { A, K, Z}
P = { A → ε | ZA | KA , K → (A) | [A] , Z → a | b | c }

Variante 2: G = {N, T, P, A} mit
N = { A, K, Z }
P = { **A ® e | Z | K | AA** , K → (A) | [A] , Z → a | b | c }

Variante 3: G = {N, T, P, A} mit
N = { **A** }
P = { **A ® e | a | b | c | (A) | [A] | AA** }

Lösung 2.4: EBNF

a) Funktion = ("succ" | "pred" | "abs").

b) S = [(A | B)].
A = ("a" | "a" B).
B = ("b" | "b" A).

Nicht-rekursive Variante: S = ["a"] { "b" "a" } ["b"].

c) C = [C] [("a" C "b" | "b" C "a")].

d) Bezeichner = Buchstabe { (Buchstabe | Ziffer | Unterstrich) }.
Buchstabe = ("a" | "b" | "c" | … | "z" | "A" | "B" | "C" | … | "Z").
Unterstrich = "_".
Ziffer = ("0" | "1" | "2" | "3" | … | "9").

Nicht-rekursive Variante:

Bezeichner = ("a" | … | "Z") { (("a" | … | "Z") | ("0" | … | "9") | "_") }.

e) GanzeZahl = (Null | [("+" | "-")] Ziffer { (Null | Ziffer) }).
Null = "0".
Ziffer = ("1" | "2" | "3" | … | "9").

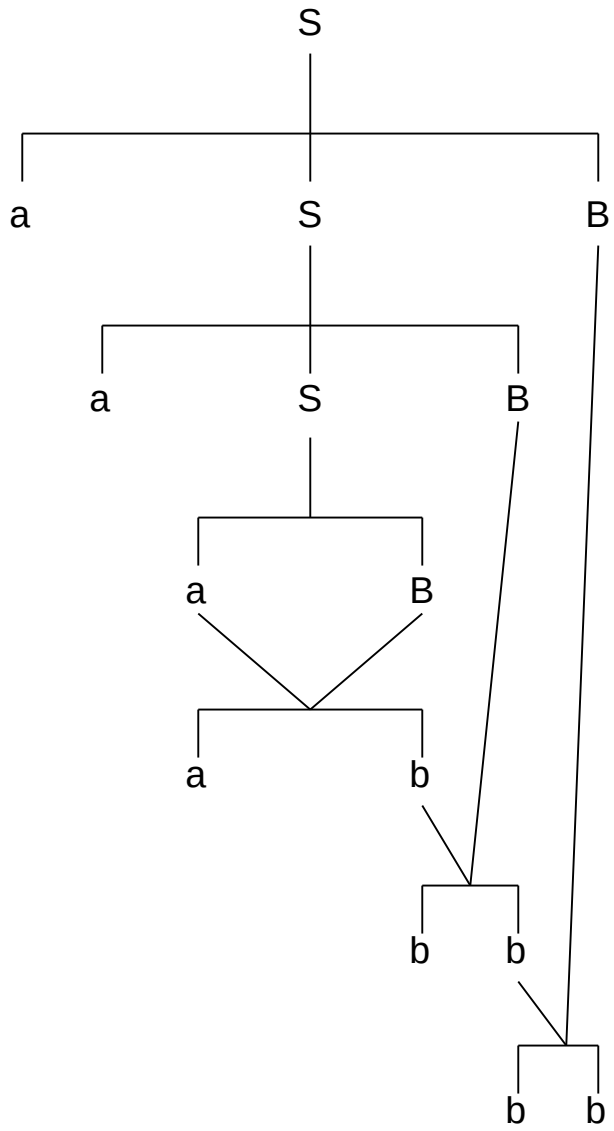
Nicht-rekursive Variante:

GanzeZahl = ("0" | [("+" | "-")] ("1" | … | "9") { ("0" | "1" | "2" | … | "9") }).

Lösungen zu Übung 3

Lösung 3.1: Grammatik

a) Der Strukturbaum für ein Wort der Länge 6:



b) Die von der Grammatik erzeugte Sprache lautet:

$L(G) = \{ a^n b^n \mid n \geq 1 \}$

c) Eine äquivalente kontextfreie Grammatik G' mit L(G) = L(G') :

G' = {N', T, P', A} mit N' = { A } und P' = { $A \rightarrow a\ A\ b$, $A \rightarrow a\ b$ }

Lösung 3.2: Algorithmus

Mit dem folgenden Algorithmus läßt sich eine Dezimalzahl in eine römische Zahl umwandeln:

```
betrachte die Zahl Z;
solange Z größer oder gleich 1000 tue
  subtrahiere 1000 von Z;
  schreibe ein M auf;
solange Z größer oder gleich 500 tue
  subtrahiere 500 von Z;
  schreibe ein D auf;
solange Z größer oder gleich 100 tue
  subtrahiere 100 von Z;
  schreibe ein C auf;
solange Z größer oder gleich 50 tue
  subtrahiere 50 von Z;
  schreibe ein L auf;
solange Z größer oder gleich 10 tue
  subtrahiere 10 von Z;
  schreibe ein X auf;
solange Z größer oder gleich 5 tue
  subtrahiere 5 von Z;
  schreibe ein V auf;
solange Z größer oder gleich 1 tue
  subtrahiere 1 von Z;
  schreibe ein I auf
```

Aufg. 3.3: Das erste und das zweite Modula-3 Programm

a) Das folgende Modula-3 Programm gibt eine Visitenkarte in der geforderten Formatierung aus:

```
MODULE Vkarte EXPORTS Main;

(* Dieses Programm erzeugt eine formatierte Visitenkarte.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 28.10.98
   Letzte Aenderung: 28.10.98
*)

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

BEGIN
  SIO.PutLine("+-----------------------------------------------------------+");
  SIO.PutLine("|                                                           |");
  SIO.PutLine("|                     Franz Kaiser                          |");
  SIO.PutLine("|                                                           |");
  SIO.PutLine("|  Finkenstr. 17                     Tel.: 07518/00349  |");
  SIO.PutLine("|  80005 Freimurg                    Fax.: 07518/00344  |");
  SIO.PutLine("|                                                           |");
  SIO.PutLine("+-----------------------------------------------------------+");
END Vkarte.
```

Die Ausgabe einer Programmausführung sollte natürlich wie auf dem Übungsblatt vorgegeben aussehen.

b) Das folgende Modula-3 Programm erzeugt aus einem zu Beginn eingegebenen Namen ein wie gefordert formatiertes Namensschild:

```
MODULE Nschild EXPORTS Main;

(* Dieses Programm erzeugt ein Namensschild fuer einen eingegebenen Namen.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 28.10.98
   Letzte Aenderung: 29.10.98
*)

IMPORT SIO;  (* Importiere vordefinierte Ein-/Ausgabe-Operationen *)
IMPORT Text; (* Importiere Prozeduren und Funktionen zur Textmanipulation *)

PROCEDURE Ausgeben(Name: TEXT)=
(* Die Prozedur gibt ein Schild mit dem als Parameter uebergebenen Namen aus. *)
BEGIN
  SIO.PutLine("+------------------------------------------------------------+");
  SIO.PutLine("|                                                            |");
  SIO.PutText("|              ");          (* Zeilenanfang mit Laenge 15 *)
  SIO.PutText(Name);
  SIO.PutText(Text.Sub("                                                |",
                       Text.Length(Name),
                       45-Text.Length(Name)(* 45 ist die Laenge der Restzeile *)
                      )
              );
  SIO.Nl();
  SIO.PutLine("|                                                            |");
  SIO.PutLine("+------------------------------------------------------------+");
END Ausgeben;

(* Hauptprogramm *)

BEGIN
  SIO.PutLine("Geben Sie den Vor- und Nachnamen ein: ");
  Ausgeben(SIO.GetLine());
END Nschild.
```

Lösungen zu Übung 4

Lösung 4.1: Syntaxdiagramme und EBNF

a) String = """ { (Character | """ """ | """") } """.

b) Block = "[" [[":" Variablename { ":" Variablename } "|"] Statements] "]".

Lösung 4.2: Algorithmus für das Nimmspiel

Strategie bei diesem Spiel ist es, jedesmal soviele Stäbchen zu nehmen, daß die verbleibende Anzahl Stäbchen eins größer ist als die nächste durch vier teilbare Zahl. Wenn einem das gelungen ist, hat man das Spiel bereits gewonnen. Das Problem hierbei ist, daß wenn auch der Gegner diese Strategie kennt, derjenige gewinnt, der die Strategie zuerst umsetzen kann:

```
frage Gegner nach Anzahl Staebchen: n;
solange n > 1 tue
  berechne j = (n+3) modulo 4;
  wenn j = 0 dann
    setze j = 1
  sonst
    teile Gegner mit "Du hast keine Chance mehr!";
  nehme j Staebchen weg (n=n-j);
  wenn n = 1 dann
    melde "Du hast verloren!"
  sonst
    frage Gegner nach Anzahl weggenommener Staebchen: j;
    berechne verbleibende Anzahl Staebchen: n=n-j;
wenn n = 1 dann
  teile Gegner mit: "Gratuliere, Du hast gewonnen!";
```

Lösung 4.3: Einfacher Taschenrechner

```
MODULE Rechner EXPORTS Main;

(* Dieses Programm simuliert einen simplen Taschenrechner.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 28.10.98  Letzte Aenderung: 05.11.98 *)

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

PROCEDURE Berechne (Op2: REAL; Operator: CHAR; Op1: REAL): REAL =
(* Berechnet Ergebnis der arithmetischen Operation: Op1 Operator Op2 *)
BEGIN
  IF (Operator = '+') THEN
    RETURN (Op1 + Op2);
  ELSIF (Operator = '-') THEN
    RETURN (Op1 - Op2);
  ELSIF (Operator = '*') THEN
    RETURN (Op1 * Op2);
  ELSIF (Operator = '/') THEN
    RETURN (Op1 / Op2);
  ELSE
    RETURN 0.0; (* Undefinierte Operation, kein Ergebnis *)
  END;
END Berechne;
```

```
BEGIN
  SIO.PutLine("Bitte geben Sie einen arithmetischen Ausdruck ein: ");
  SIO.PutReal(Berechne(SIO.GetReal(), SIO.GetChar(), SIO.GetReal()));
  SIO.PutLine(" lautet das Ergebnis.");
END Rechner.
```

Bei diesem Programm ist zu beachten, daß in Modula-3 nicht festgelegt ist, in welcher Reihenfolge die übergebenen aktuellen Parameter einer Funktion auszuwerten sind. Während diese Auswertung bei den Unix-Installationen von SRC Modula-3 von links nach rechts erfolgt, geschieht diese unter Windows (zumindest in Verbindung mit Visual C++) von rechts nach links. Auswertungsreihenfolge der aktuellen Parameter unter:

Unix	**1.**	2.	**3.**
Windows	**3.**	2.	**1.**

```
Berechne(SIO.GetReal(), SIO.GetChar(), SIO.GetReal())
```

Diese Tatsache ist unerheblich, solange sämtliche Funktionen, die als Parameter dienen, von Seiteneffekten frei sind. Leider ist dies im obigen Programm nicht der Fall, da das Lesen von der Tastatur stets den Seiteneffekt besitzt, daß der Eingabepuffer um die gelesenen Zeichen verkürzt wird. Während also unter Unix als erster Parameter die zuerst eingegebene REAL-Zahl übergeben wird, ist dies unter Windows die zweite eingegebene REAL-Zahl, da dort der Funktionsaufruf für den ersten Parameter zuletzt ausgewertet wird. Dies läßt sich jedoch leicht, wie oben geschehen, durch Vertauschen der formalen Parameter berichtigen.

Die Ausgabe eines Programmlaufs (**mit nachträglich eingefügten Eingabewerten**) sieht so aus:

```
Bitte geben Sie einen arithmetischen Ausdruck ein:
6.7-5.6
1.0999999 lautet das Ergebnis.
```

Lösung 4.4: Gewicht einer Hohlkugel

```
MODULE MantelGewicht EXPORTS Main;

(* Dieses Programm berechnet das Gewicht einer Hohlkugel.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 05.11.98  Letzte Aenderung: 05.11.98 *)

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

(* Funktionen mit konstantem Wert *)

PROCEDURE VierDrittel(): REAL =
(* Gibt den konstanten Wert 4/3 zurueck *)
BEGIN
  RETURN 4.0/3.0;
END VierDrittel;

PROCEDURE VierDrittelPi(): REAL =
(* Gibt den konstanten Wert 4/3*Pi zurueck *)
BEGIN
  RETURN VierDrittel()*3.141592653; (* 4/3*Pi *)
END VierDrittelPi;

(* Funktionen fuer die Berechnung *)

PROCEDURE Quadrat(x: REAL): REAL =
(* Berechnet das Quadrat der Zahl x *)
BEGIN
  RETURN x*x;
END Quadrat;
```

```
PROCEDURE Kubik(x: REAL): REAL =
(* Berechne den Wert von x hoch 3 *)
BEGIN
  RETURN x*Quadrat(x);
END Kubik;

PROCEDURE VolumenKugel(r: REAL): REAL =
(* Berechnet das Volumen einer Kugel mit Radius r *)
BEGIN
  RETURN VierDrittelPi()*Kubik(r);
END VolumenKugel;

PROCEDURE VolumenMantel(volaussen: REAL; volinnen: REAL): REAL =
(* Berechnet das Volumen des Raumes zwischen Aussen- und Innenkoerper *)
BEGIN
  IF ((volaussen - volinnen) < 0.0) THEN
    (* Falls Aussen- kleiner als Innenkoerper, gib 0 zurueck *)
    RETURN 0.0;
  ELSE
    (* Sonst gib den Wert zurueck *)
    RETURN volaussen - volinnen;
  END;
END VolumenMantel;

(* Funktionen fuer die Eingabe *)

PROCEDURE LeseWertEin(Frage: TEXT): REAL =
(* Gibt den Text von Frage aus und liest eine REAL-Zahl als Antwort ein *)
BEGIN
  SIO.PutText(Frage);
  RETURN SIO.GetReal();
END LeseWertEin;

(* Hauptprogramm *)

BEGIN
  SIO.PutReal(
    VolumenMantel(
                  VolumenKugel(LeseWertEin("Geben Sie den Radius (in cm) der
                                            aeusseren Kugel ein: ")),
                  VolumenKugel(LeseWertEin("Geben Sie den Radius (in cm) der
                                            inneren Kugel ein: "))
                )
    * LeseWertEin("Geben Sie die Dichte (in g/cm^3) des Materials ein: ")
              );
  SIO.PutLine(" g ist das Gewicht der Hohlkugel.");
END MantelGewicht.
```

Die Ausgabe eines Programmlaufs (**mit nachträglich eingefügten Eingabewerten**) sieht so aus:

```
Geben Sie den Radius (in cm) der inneren Kugel ein:  5.9
Geben Sie den Radius (in cm) der aeusseren Kugel ein:  6.0
Geben Sie die Dichte (in g/cm^3) des Materials ein:  1.8
80.08033 g ist das Gewicht der Hohlkugel.
```

Lösungen zu Übung 5

Lösung 5.1: Rekursiver Primzahltest

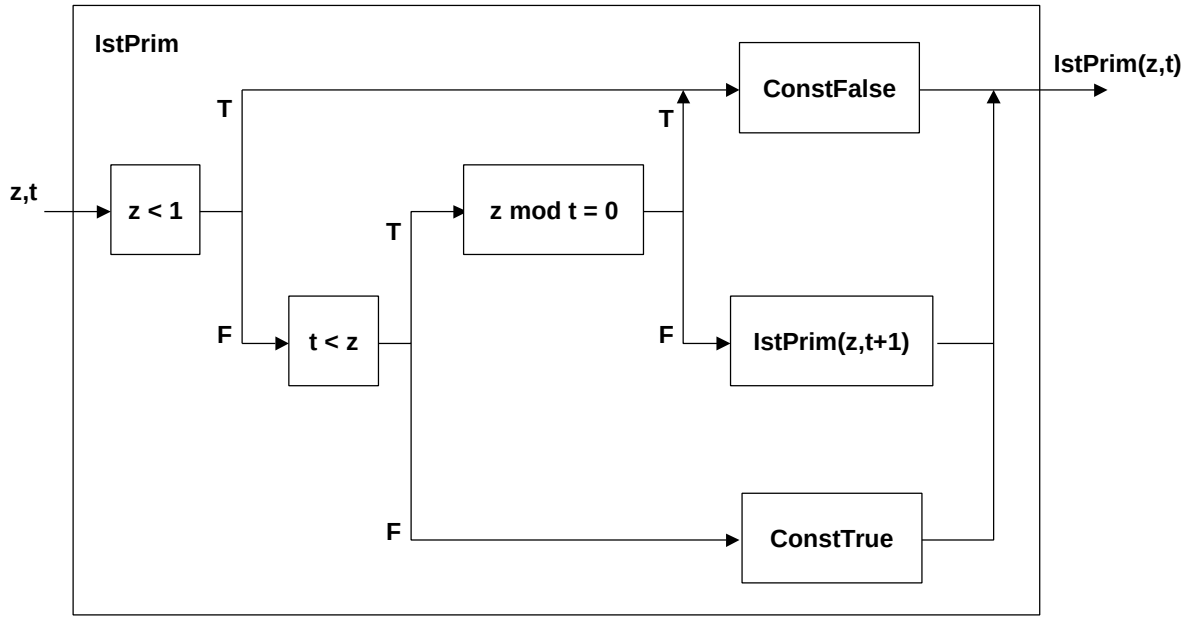
a) Hauptprogramm mit Primzahltestfunktion IstPrim:

```
MODULE Prim51 EXPORTS Main;

(* Dieses Programm prueft mittels Rekursion, ob eine eingegebene Zahl eine
   Primzahl ist (ohne Optimierung).
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 11.11.98  Letzte Aenderung: 11.11.98 *)

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

PROCEDURE IstPrim(z: INTEGER; t: INTEGER): BOOLEAN =
(*  Prueft, ob z durch t teilbar ist, und fuehrt eine Rekursion mit t+1 durch,
    wenn das nicht der Fall ist *)
BEGIN
  (* Pruefe, ob z groesser 1 *)
  IF (z <= 1) THEN
    RETURN FALSE; (* Negative Zahl, die Null und die Eins :) sind keine Primzahlen *)
  END;

  IF (t < z) THEN             (* Abbruchkriterium fuer Rekursion *)
    IF ((z MOD t) = 0) THEN   (* Pruefe, ob z durch t teilbar *)
      RETURN FALSE;
    ELSE
      RETURN IstPrim(z, t+1); (* Rekursion mit t+1 *)
    END;
  ELSE
    RETURN TRUE; (* Abbruchkriterium erfuellt, ohne Teiler zu finden *)
  END;
END IstPrim;

BEGIN
  SIO.PutText("7 ist eine Primzahl: ");
  SIO.PutBool(IstPrim(7, 2));           (* Beginne mit Pruefung bei 2 *)
  SIO.Nl();
END Prim51.
```

Die Ausgabe für die verlangten vier Zahlen sieht dann folgendermaßen aus:

```
  7 ist eine Primzahl: TRUE
 12 ist eine Primzahl: FALSE
135 ist eine Primzahl: FALSE
277 ist eine Primzahl: TRUE
```

b) Funktionsnetz für die rekursive Primzahltestfunktion IstPrim:



Lösung 5.2: Rekursive Modulo-Funktion

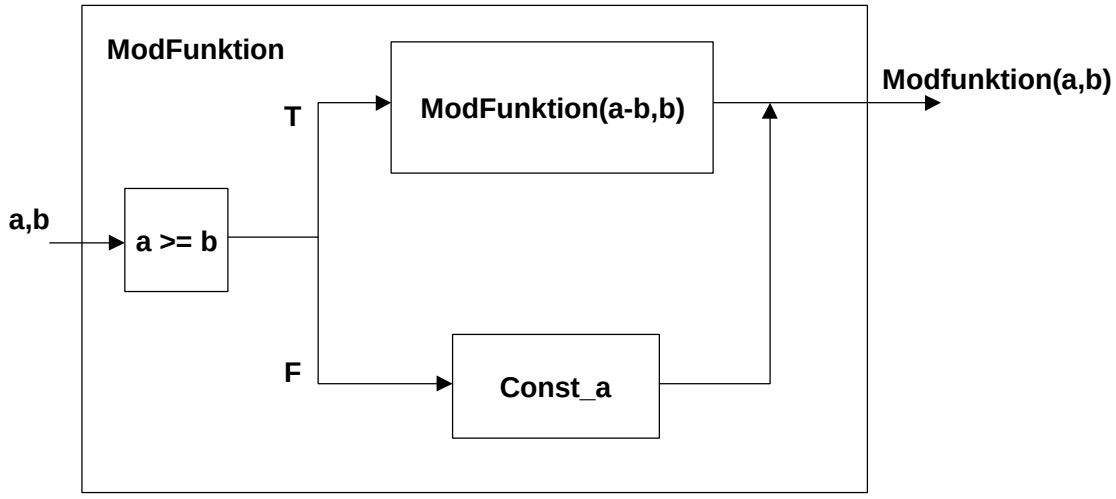
a) Hauptprogramm mit rekursiver Modulo-Funktion ModFunktion:

```
MODULE Modulo52 EXPORTS Main;

(* Dieses Programm berechnet rekursiv die Funktion a mod b fuer a >= 0 und b >= 0.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 11.11.98    Letzte Aenderung: 11.11.98 *)

IMPORT SIO; (* Importiere vordefinierte Ein-/Ausgabefunktionen *)

PROCEDURE ModFunktion(a: INTEGER; b: INTEGER): INTEGER =
(* Bestimmt ModFunktion(a-b, b) fuer a >= b und a fuer a < b  *)
BEGIN
  (* Fuer b = 0 ist die Modulo-Operation nicht definiert *)
  IF (b = 0) THEN
    SIO.PutLine("ModFunktion: Division durch 0 nicht zulaessig!");
    RETURN a;  (* Gib irgendetwas, zum Beispiel Rest a zurueck *)
  END;

  IF (a >= b) THEN
    RETURN ModFunktion(a-b, b);
  ELSE
    RETURN a;
  END;
END ModFunktion;

BEGIN
  SIO.PutText("10 mod 3 = ");
  SIO.PutInt(ModFunktion(10, 3));
  SIO.Nl();
END Modulo52.
```

Die Ausgabe für die erforderlichen Zahlenpaare sieht wie folgt aus:

```
10 mod 3 = 1
19 mod 4 = 3
121 mod 11 = 0
129 mod 7 = 3
```

b) Funktionsnetz für die rekursive Modulo-Funktion ModFunktion:



Lösung 5.3: Umkehren eines Texts mittels Rekursion

```
MODULE Reverse53 EXPORTS Main;

(* Dieses Program dreht einen Text mit Hilfe einer rekursiven Funktion um.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 11.11.98   Letzte Aenderung: 11.11.98 *)

IMPORT SIO;  (* Importiere Funktionen fuer Ein-/Ausgabe *)
IMPORT Text; (* Importiere Funktionen/Prozeduren fuer Texte *)

PROCEDURE Reverse (t: TEXT): TEXT =
(* Dreht einen Text mittels Rekursion um *)
BEGIN
  (* Breche Rekursion ab, wenn Text in Parameter t leer *)
  IF NOT Text.Empty(t) THEN
    RETURN (  Reverse (Text.Sub(t,1)) (* Rekursion mit um 1 verkuerzten Text *)
            & Text.FromChar(Text.GetChar(t,0)) );
  ELSE
    RETURN (""); (* Gib bei Abbruch leeren Text zurueck *)
  END;
END Reverse ;

BEGIN
  SIO.PutLine("Geben Sie den umzukehrenden Text ein: ");
  (* Lese Text ein, drehe ihn um und gib ihn wieder aus *)
  SIO.PutLine(Reverse(SIO.GetLine()));
END Reverse53.
```

Ein protokollierter Programmlauf (**mit nachträglich eingefügter Eingabe**) sieht dann so aus:

```
Geben Sie den umzukehrenden Text ein:
Dieser Text wird umgekehrt wieder ausgegeben.
.nebegegsua redeiw trhekegmu driw txeT reseiD
```

Aufg. 5.4: Chiffrierung

a) Hauptprogramm mit Chiffrierfunktion Chiffriere:

```
MODULE Chiffre54a EXPORTS Main;

(* Dieses Programm fuehrt eine Chiffrierung von Zeichen durch.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 11.11.98   Letzte Aenderung: 11.11.98 *)

IMPORT SIO; (* Importiere vordefinierte Ein-/Ausgabefunktionen *)

PROCEDURE Ersetze(c: CHAR; index: INTEGER;
                  start: CHAR; laenge: INTEGER): CHAR =
(* Ersetze Zeichen c durch das um Index verschobene Zeichen modulo die
   Laenge des Wertebereichs, der beim Zeichen start beginnt. *)
BEGIN
  RETURN VAL(((((ORD(c)+index)-ORD(start)) MOD laenge)+ORD(start)), CHAR);
END Ersetze;


PROCEDURE Chiffriere(c: CHAR): CHAR =
(* Gibt fuer Grossbuchstaben den dritten Nachfolger,
        fuer Kleinbuchstaben den fuenften Nachfolger,
        fuer Ziffern den zweiten Vorgaenger und
        ansonsten dasselbe Zeichen aus  *)
BEGIN
  IF (ORD('A') <= ORD(c)) AND (ORD(c) <= ORD('Z')) THEN
    RETURN Ersetze(c, 3, 'A', 26);
  ELSIF (ORD('a') <= ORD(c)) AND (ORD(c) <= ORD('z')) THEN
    RETURN Ersetze(c, 5, 'a', 26);
  ELSIF (ORD('0') <= ORD(c)) AND (ORD(c) <= ORD('9')) THEN
    RETURN Ersetze(c, -2, '0', 10);
  ELSE
    RETURN c;
  END;
END Chiffriere;

BEGIN
  SIO.PutText("b = ");
  SIO.PutChar(Chiffriere('b'));
  SIO.Nl();
  ...
  SIO.PutText("Programmierung - Info 1 = ");
  SIO.PutChar(Chiffriere('P')); SIO.PutChar(Chiffriere('r'));
  SIO.PutChar(Chiffriere('o')); SIO.PutChar(Chiffriere('g'));
  SIO.PutChar(Chiffriere('r')); SIO.PutChar(Chiffriere('a'));
  SIO.PutChar(Chiffriere('m')); SIO.PutChar(Chiffriere('m'));
  SIO.PutChar(Chiffriere('i')); SIO.PutChar(Chiffriere('e'));
  SIO.PutChar(Chiffriere('r')); SIO.PutChar(Chiffriere('u'));
  SIO.PutChar(Chiffriere('n')); SIO.PutChar(Chiffriere('g'));
  SIO.PutChar(Chiffriere(' ')); SIO.PutChar(Chiffriere('-'));
  SIO.PutChar(Chiffriere(' ')); SIO.PutChar(Chiffriere('I'));
  SIO.PutChar(Chiffriere('n')); SIO.PutChar(Chiffriere('f'));
  SIO.PutChar(Chiffriere('o')); SIO.PutChar(Chiffriere(' '));
  SIO.PutChar(Chiffriere('1')); SIO.Nl();
END Chiffre54a.
```

Ein Programmlauf ergibt dann folgendes Ergebnis:

```
b = g
K = N
z = e
Y = B
5 = 3
1 = 9
+ = +
Programmierung - Info 1 = Swtlwfrrnjwzsl - Lskt 9
```

b) Analog zu Aufg. 5.4 a) ergibt sich:

```
MODULE Chiffre54b EXPORTS Main;

(* Dieses Program dreht einen Text mit Hilfe einer rekursiven Funktion um
   und chiffriert ihn gleichzeitig.
   ... *)

...

PROCEDURE Ersetze(c: CHAR; index: INTEGER;
                  start: CHAR; laenge: INTEGER): CHAR =
...
END Ersetze;


PROCEDURE Chiffriere(c: CHAR): CHAR =
...
END Chiffriere;

PROCEDURE ChiffUmkehr (t: TEXT): TEXT =
(* Kehrt einen Text mittels Rekursion um und chiffriert ihn gleichzeitig *)
BEGIN
  (* Breche Rekursion ab, wenn Text in Parameter t leer *)
  IF NOT Text.Empty(t) THEN
    RETURN (  ChiffUmkehr (Text.Sub(t,1)) (* Rekursion mit um 1 verkuerzten Text *)
            & Text.FromChar(Chiffriere(Text.GetChar(t,0))) );
  ELSE
    RETURN (""); (* Gib bei Abbruch leeren Text zurueck *)
  END;
END ChiffUmkehr;

BEGIN
  SIO.PutLine("Geben Sie den umzukehrenden und zu chiffrierenden Text ein: ");
  (* Lese Text ein, kehre ihn um und chiffriere ihn dabei, dann gib ihn wieder aus *)
  SIO.PutLine(ChiffUmkehr(SIO.GetLine()));
END Chiffre54b.
```

Ein Programmlauf liefert dann folgende Ausgabe (**mit nachträglich eingefügter Eingabe**):

```
Geben Sie den umzukehrenden und zu chiffrierenden Text ein:
Dieser Text wird umgedreht und gleichzeitig auch chiffriert.
.ywjnwkknmh mhzf lnynjemhnjql isz ymjwijlrz iwnb ycjW wjxjnG
```

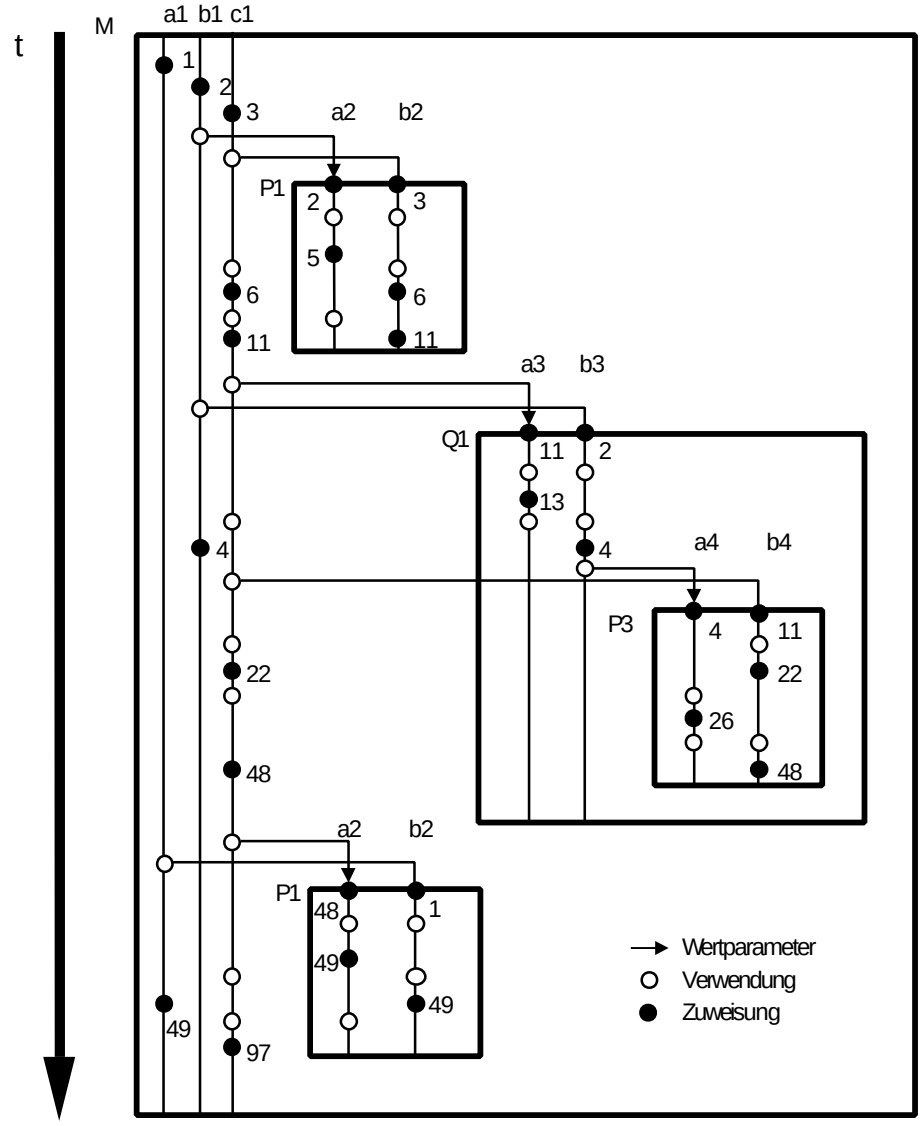
Lösungen zu Übung 6

Lösung 6.1: Gültigkeitsbereich und Lebensdauer

a) 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 P3( a4: INTEGER; VAR b4: INTEGER)=
  BEGIN
    c1 := c1 + b4;
    a4 := a4 + c1;
    b4 := b4 + a4
  END P3;

  (* Ende Gültigkeitsbereich 4 *)

BEGIN
  a3 := a3 + b3;
  b3 := b3 + a3 - c1;
  P3( b3, c1)
END Q1;

(* Ende Gültigkeitsbereich 3 *)

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

b) Ablaufdiagramm mit der Lebensdauer sowie Zwischen- und Endwerten der Variablen:



Lösung 6.2: Flagge Alphanumericas

```
MODULE Alpha62 EXPORTS Main;

(* Das Programm berechnet die Flagge Alphanumericas in verschiedenen Groessen.
   Autoren         : Oliver Meyer, Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 12.11.98  Letzte Aenderung: 19.11.98 *)

IMPORT SIO; (* Importiere Ein-/Ausgabefunktionen *)

PROCEDURE SchreibeFlagge (groesse: CARDINAL) =
(* Erzeugt die Flagge Alphanumericas. Die Anzahl Zeichen pro Zeichengruppe
   in einer Zeile wird durch Zweierpotenzen bestimmt. Die erste Zeile
   hat die Anzahl groesse, die zweite groesse/2, die dritte groesse/4 usw. *)

VAR zeichenzahl: CARDINAL; (* Anzahl Zeichen pro Zeichengruppe in einer Zeile *)

BEGIN
  zeichenzahl := groesse;   (* Initialisierung fuer 1. Zeile mit groesse *)

  WHILE zeichenzahl >= 1 DO

    (* Durchlaufe Schleife bis Anzahl Zeichen pro Gruppe < 1 (also 0) *)
    FOR gruppe := 1 TO (groesse DIV zeichenzahl) DO
      (* Gib fuer jede Gruppe die notwendige Anzahl Zeichen aus *)
      FOR minus := 1 TO zeichenzahl DO SIO.PutText("-"); END;
      FOR stern := 1 TO zeichenzahl DO SIO.PutText("*"); END;
    END;
    SIO.Nl();

    (* Berechne Anzahl Zeichen pro Gruppe in naechster Zeile *)
    zeichenzahl := zeichenzahl DIV 2;
  END;
END SchreibeFlagge;

VAR flaggengroesse: INTEGER;

BEGIN
  SIO.PutText("Geben Sie die gewuenschte Flaggengroesse ein (2,4,8,16,32): ");
  flaggengroesse := SIO.GetInt();

  (* Erzeuge nur eine Flagge fuer die Groessen: 2, 4, 8, 16 und 32 *)
  CASE flaggengroesse OF
    2,4,8,16,32 => SIO.PutLine("Hier die gewuenschte Flagge: ");
                   SIO.Nl();
                   SchreibeFlagge(flaggengroesse);
  ELSE
    SIO.PutLine("Dafuer gibts keine Flagge!");
  END;

END Alpha62.
```

Ein Programmlauf erzeugt dann folgende Ausgabe (**mit nachträglich eingefügter Eingabe**):

```
Bitte geben Sie die gewuenschte Flaggengroesse ein (2,4,8,16,32):  16
Hier die gewuenschte Flagge:

----------------**************** 
--------********--------******** 
----****----****----****----**** 
--**--**--**--**--**--**--**--** 
-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*
```

Lösung 6.3: Schleifenkonversion

Im folgenden sind nur die zu realisierenden Prozeduren nochmals aufgeführt:

```
PROCEDURE MitWhile (n: CARDINAL): CARDINAL =
(* Umsetzung mit einer WHILE-Schleife *)

VAR fakultaet: CARDINAL;   (* Rueckgabewert der Funktion *)
    i: CARDINAL;           (* Laufvariable fuer die WHILE-Schleife *)

BEGIN
  fakultaet := 1; i := 2;  (* Initialisierung *)
  WHILE i <= n DO
    fakultaet := fakultaet * i;
    i := i + 1;
  END;
  RETURN fakultaet;
END MitWhile;

PROCEDURE MitRepeat (n: CARDINAL): CARDINAL =
(* Umsetzung mit einer REPEAT-Schleife *)

VAR fakultaet: CARDINAL;   (* Rueckgabewert der Funktion *)
    i: CARDINAL;           (* Laufvariable fuer die REPEAT-Schleife *)

BEGIN
    fakultaet := 1; i := 2;      (* Initialisierung *)

    (* Die Repeat-Schleife laeuft immer mindestens einmal durch.  Daher
       wird die Schleife nur dann gestartet, wenn es auch wirklich
       mindestens einen Durchlauf gibt. Eigentlich untaugliche Lösung! *)
    IF n >= i THEN
      REPEAT
        fakultaet:=fakultaet*i;
        i := i+1;
      UNTIL i > n;
    END;
    RETURN fakultaet;
END MitRepeat;

PROCEDURE MitLoop (n: CARDINAL): CARDINAL =
(* Umsetzung mit eine LOOP-Schleife *)

VAR fakultaet: CARDINAL;   (* Rueckgabewert der Funktion *)
    i: CARDINAL;           (* Laufvariable fuer die LOOP-Schleife *)

BEGIN
  fakultaet := 1; i := 2;  (* Initialisierung *)
  LOOP
    IF i > n THEN
      EXIT;
    ELSE
      fakultaet := fakultaet * i;
      i := i + 1;
    END;
  END;
  RETURN fakultaet;
END MitLoop;
```

Lösung 6.4: Implementierung römischer Zahlen

```
MODULE Rome64 EXPORTS Main;

(* Dieses Programm ermittelt die roemische Darstellung zu einer positiven,
   ganzen Dezimalzahl.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 19.11.98  Letzte Aenderung: 19.11.98 *)

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

VAR zahl: INTEGER;

PROCEDURE GibSymbol(wert: INTEGER): CHAR =
(* Gibt das zur Zahl wert gehoerige roemische Zahlensymbol zurueck *)
VAR resultat: CHAR;

BEGIN
  CASE wert OF
    1000 => resultat := 'M'; |
    500  => resultat := 'D'; |
    100  => resultat := 'C'; |
    50   => resultat := 'L'; |
    10   => resultat := 'X'; |
    5    => resultat := 'V'; |
    1    => resultat := 'I';
  ELSE
    resultat := '-'; (* Es existiert kein Symbol fuer den Wert! *)
  END;
  RETURN resultat;
END GibSymbol;

PROCEDURE BerechneRoemisch(zahl: CARDINAL): TEXT =
(* Berechnet die roemische Zahlendarstellung unter Verwendung zweier Schleifen.
   Die aeussere Schleife wird dabei durch den Wert des aktuellen Zahlensymbols
   (M = 1000, D = 500, C = 100, ...) gesteuert, die innere ermittelt die
   notwendigen Zahlensymbole pro Wert *)

VAR roemischzahl: TEXT;    (* Variable fuer das Resultat *)
    wert        : INTEGER; (* Aktueller Wert des roemischen Zahlensymbols *)
    teiler      : INTEGER; (* Hilfsvariable, um Wert des Symbols der naechsten
                              Iteration zu bestimmen *)
BEGIN
  roemischzahl := ""; (* Initialisiere mit roemisch Null *)
  wert := 1000;       (* Starte mit Wert von M = 1000 *)
  teiler := 2;        (* Teiler, um Wert fuer naechste Iteration zu berechnen *)

  WHILE (wert >= 1) DO
    (* Durchlaufe Schleife von M = 1000 bis I = 1 *)

    WHILE (zahl >= wert) DO
      (* Gib alle in Zahl enthaltenen Symbole mit Wert wert aus *)
      zahl := zahl-wert;
      roemischzahl := roemischzahl & Text.FromChar(GibSymbol(wert));
    END;

    wert := wert DIV teiler;  (* Berechne Wert fuer naechste Iteration *)
```

```
    (* Bestimme Wert der Hilfsvariablen teiler fuer naechste Iteration.
       Es gilt: 500 = 1000/2, 100 = 500/5, 50 = 100/2, 10 = 50/5  etc. *)
    IF (teiler = 2) THEN
      teiler := 5;
    ELSE
      teiler := 2;
    END;

  END;

  RETURN roemischzahl;
END BerechneRoemisch;

BEGIN
  SIO.PutText("Geben Sie die in roemische Darstellung zu wandelnde Zahl ein: ");
  zahl := SIO.GetInt();

  IF (zahl >= 0) THEN
    (* Wandle positive Zahl/Null in roemische Zahl und gib das Ergebnis aus *)
    SIO.PutLine("Roemische Zahl: " & BerechneRoemisch(zahl));
  ELSE
    (* Negative Zahl wurde eingegeben *)
    SIO.PutLine("Dafuer gibt es keine roemische Darstellung!");
  END;

END Rome64.
```

Ein Programmlauf erzeugt dann die folgende Ausgabe (**mit nachträglich eingefügter Eingabe**):

```
Geben Sie die in roemische Darstellung zu wandelnde Zahl ein:  37
Roemische Zahl: XXXVII
```

Lösungen zu Übung 7

Lösung 7.1: Datenstrukturen für

a) Datum mit Uhrzeit:

```
Uhrzeit = RECORD
            sekunde: [0..59];
            minute : [0..59];
            stunde : [0..23];
          END;

Datum = RECORD
          tag  : [1..31];
          monat: [1..12];
          jahr : [1800..3000];
        END;

UhrzeitDatum = RECORD
                 uhrzeit: Uhrzeit;
                 datum  : Datum;
               END;
```

b) Daten einer Person:

```
Name = RECORD
         vorname : ARRAY [1..25] OF CHAR;
         nachname: ARRAY [1..40] OF CHAR;
       END;

Adresse = RECORD
            strasse: ARRAY [1..20] OF CHAR;
            hausNr : [1..1000];
            plz    : [0..99999];
            ort    : ARRAY [1..20] OF CHAR;
          END;

Familienstand = {Ledig, Verheiratet, Geschieden, Verwitwet};

Person = RECORD
           name         : Name;
           adresse      : Adresse;
           gebDatum     : Datum;
           beruf        : ARRAY [1..20] OF CHAR;
           familienstand: Familienstand;
         END;
```

c) ein Bankkonto:

```
KontoArt = {Girokonto, Sparkonto, Anlagekonto};

Status   = {Frei, Gesperrt};

Konto    = RECORD
             kontoNr   : CARDINAL;
             kontoArt  : KontoArt;
             inhaber   : ARRAY [1..10] OF Person;
             verzinsung: [0..10];  (* in Prozent *)
             saldo     : INTEGER;
             status    : Status;
           END;
```

d) die Daten eines Angestellten:

```
Steuerklasse = {I, II, III, IV, V, VI};

Angestellter = RECORD
                 person      : Person;
                 eintritt    : Datum;
                 gehalt      : [1000..100000];
                 kontoNr     : CARDINAL; (* fuer das Gehalt *)
                 steuerklasse: Steuerklasse;
               END;
```

e) einen Einkaufszettel:

```
Position = RECORD
             anzahl : CARDINAL;
             artikel: ARRAY [1..30] OF CHAR;
           END;

Anzahlposition = [1..100];

Einkaufszettel = RECORD
                   anzahlPositionen: Anzahlposition;
                   position        : ARRAY Anzahlposition OF Position;
                 END;
```

f) den Bücherbestand einer Bibliothek:

```
Buch = RECORD
         autor     : Name;
         titel     : ARRAY [1..50] OF CHAR;
         verlag    : ARRAY [1..30] OF CHAR;
         isbn      : CARDINAL;
         erschienen: [0..3000];
       END;

InventarBuch = RECORD
                 buch    : Buch;
                 erworben: [1800..3000];
                 signatur: ARRAY [1..10] OF CHAR;
               END;

Bestand = ARRAY [0..1000] OF InventarBuch;
```

Lösung 7.2: Lottomanisches Lotto

```
MODULE Lotto72 EXPORTS Main;

(* Dieses Programm dient der Ein- und Ausgabe lottomanischer Lottozettel.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 26.11.98  Letzte Aenderung: 27.11.98 *)


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


CONST MinAnzTips   =  0;
      MaxAnzTips   =  6;
      MinAnzSpiele =  0;
      MaxAnzSpiele = 10;
      MinTip       =  1;
      MaxTip       = 49;


TYPE Tip    = [MinTip..MaxTip];

     Spiel  = RECORD
                anzahlTips: [MinAnzTips..MaxAnzTips];
                tips      : ARRAY [MinAnzTips+1..MaxAnzTips] OF Tip;
              END;

     Zettel = RECORD
                anzahlSpiele: [MinAnzSpiele..MaxAnzSpiele];
                spiele      : ARRAY [MinAnzSpiele+1..MaxAnzSpiele] OF Spiel;
              END;


VAR zettel       : Zettel;
    zettelFuellen: BOOLEAN; (* Steuervariable *)


(* Prozeduren fuer die Bearbeitung eines Lottozettels *)

PROCEDURE InitZettel(VAR zettel: Zettel)=
(* Initialisiert einen Lottozettel mit den Startwerten:
   Kein Spiel und keine Tips *)

BEGIN
  zettel.anzahlSpiele := 0;         (* Kein Spiel gemacht *)

  WITH spiele = zettel.spiele DO
    FOR i := FIRST(spiele) TO LAST(spiele) DO
      spiele[i].anzahlTips := 0;    (* Keine Tips *)
    END;
  END;
END InitZettel;
```

```
PROCEDURE FuelleSpielAus(VAR spiel: Spiel)=
(* Fuellt die Tips des uebergebenen Spiels aus *)

VAR tipZahl     : [MinAnzTips..MaxAnzTips]; (* Anzahl abgegebener Tips *)
    ihrTip      : INTEGER;
    weiterTippen: BOOLEAN;                  (* Steuervariablen *)
    tipOK       : BOOLEAN;

BEGIN
  tipZahl := 0; weiterTippen := TRUE;
  REPEAT

    (* Bitte Benutzer um Eingabe des naechsten Tips *)

    SIO.PutText("Geben Sie Ihren "); SIO.PutInt(tipZahl + 1);
    SIO.PutText(". Tip (0 = Ende): ");
    ihrTip := SIO.GetInt();


    IF (ihrTip >= FIRST(Tip) ) AND ( ihrTip <= LAST(Tip) ) THEN

      (* Abgegebener Tip im zulaessigen Bereich bspw. [1..49] *)

      (* Stelle sicher, dass die Zahl im Spiel nicht bereits getippt wurde *)
      tipOK := TRUE;
      FOR i := FIRST(spiel.tips) TO spiel.anzahlTips DO
        IF (spiel.tips[i] = ihrTip) THEN tipOK := FALSE; END;
      END;

      IF tipOK THEN

        (* Tip in Ordnung, kann in Spiel uebernommen werden *)
        tipZahl := tipZahl + 1;
        spiel.tips[tipZahl] := ihrTip;
        spiel.anzahlTips := tipZahl;

      ELSE

        (* Zahl wurde bereits getippt *)
        SIO.PutText("Die Zahl "); SIO.PutInt(ihrTip);
        SIO.PutLine(" wurde bereits getippt!");

      END;

    ELSIF (ihrTip = 0) THEN
      weiterTippen := FALSE; (* Tippen beenden, da 0 eingegeben wurde. *)
    ELSE

      (* Tip liegt nicht innerhalb des zulaessigen Bereichs *)
      SIO.PutText("Die Zahl "); SIO.PutInt(ihrTip);
      SIO.PutLine(" ist kein gueltiger Tip!");

    END;
  UNTIL (tipZahl = LAST(spiel.tips) ) OR NOT weiterTippen;

  SIO.Nl(); (* Gehe in neue Zeile *)
END FuelleSpielAus;
```

```
PROCEDURE FuelleZettelAus(VAR zettel: Zettel)=
(* Fuellt den gegebenen Lottozettel aus *)

VAR spielZahl     : [MinAnzSpiele..MaxAnzSpiele]; (* Anzahl kompletter Spiele *)
    weiterSpielen : BOOLEAN;

BEGIN

  (* Initialisiere Parameter zettel mit leerem Lottozettel *)
  InitZettel(zettel);

  (* Beginne mit Ausfuellen des Zettels *)
  SIO.PutLine("Geben Sie bitte die Daten zum Lottozettel ein: "); SIO.Nl();

  spielZahl := 0; weiterSpielen := TRUE;
  REPEAT

    (* Lasse ein Spiel nach dem anderen ausfuellen *)
    SIO.PutText("Spiel Nr. ");
    SIO.PutInt(spielZahl + 1);
    SIO.PutLine(": ");
    SIO.Nl();
    FuelleSpielAus(zettel.spiele[spielZahl + 1]);

    IF (zettel.spiele[spielZahl + 1].anzahlTips >= 1) THEN

      (* Wenigstens ein Tip wurde abgegeben, also erhoehe Anzahl der Spiele *)
      spielZahl := spielZahl + 1;
      zettel.anzahlSpiele := spielZahl;

    END;

    (* Frage Benutzer, ob er noch weiterspielen moechte *)
    IF (spielZahl < LAST(zettel.spiele)) THEN
      weiterSpielen := StelleFrage("Wollen Sie weiterspielen"); SIO.Nl();
    END;

  UNTIL (spielZahl = LAST(zettel.spiele) ) OR NOT weiterSpielen;
END FuelleZettelAus;


PROCEDURE DruckeZettel(VAR zettel: Zettel)=
(* Druckt den gegebenen Lottozettel *)

BEGIN
  SIO.PutLine("Ihr Lottozettel enthaelt folgende Daten: "); SIO.Nl();

  (* Gib alle Spiele der Reihe nach aus *)
  FOR spiel := FIRST(zettel.spiele) TO zettel.anzahlSpiele DO

    SIO.PutText("Spiel Nr. "); SIO.PutInt(spiel); SIO.PutText(": ");

    (* Gib alle Tips des Spiels der Reihe nach aus *)
    FOR tip := FIRST(zettel.spiele[spiel].tips) TO
        zettel.spiele[spiel].anzahlTips
    DO
      SIO.PutText("  ");
      SIO.PutInt(zettel.spiele[spiel].tips[tip]);
    END;

    SIO.Nl();
  END;
END DruckeZettel;
```

```
(* Hilfsprozedur *)

PROCEDURE StelleFrage(frage: TEXT): BOOLEAN=
(* Zeigt den als Parameter uebergebenen Fragetext an und prueft,
   ob der Benutzer mit j oder J (ja) oder sonstiges (nein) antwortet. *)
VAR resultat: BOOLEAN;
    antwort : CHAR;

BEGIN
  (* Leere Eingabepuffer!! Sonst wird vorhergehendes RETURN eingelesen! *)
  WHILE SIO.Available() DO
    antwort := SIO.GetChar()
  END;
  (* Zeige Fragetext an und lese Antwort *)
  SIO.PutText(frage & " (j,n)? ");
  antwort := SIO.GetChar();

  (* Werte die Antwort aus *)
  IF (antwort = 'j') OR (antwort = 'J') THEN
    resultat := TRUE;
  ELSE
    resultat := FALSE;
  END;
  RETURN resultat;
END StelleFrage;

BEGIN
  REPEAT
    FuelleZettelAus(zettel); SIO.Nl();
    DruckeZettel(zettel); SIO.Nl();
    (* Frage Benutzer, ob er noch einen Zettel ausfuellen will *)
    zettelFuellen := StelleFrage("Weiteren Zettel ausfuellen");
  UNTIL NOT zettelFuellen;
END Lotto72.
```

Ein Dialog mit dem Programm sieht dann so aus (**mit Benutzereingaben**):

```
Geben Sie bitte die Daten zum Lottozettel ein:

Spiel Nr. 1:

Geben Sie Ihren 1. Tip (0 = Ende): 5
Geben Sie Ihren 2. Tip (0 = Ende): 8
Geben Sie Ihren 3. Tip (0 = Ende): 17
Geben Sie Ihren 4. Tip (0 = Ende): 34
Geben Sie Ihren 5. Tip (0 = Ende): 45
Geben Sie Ihren 6. Tip (0 = Ende): 0

Wollen Sie weiterspielen (j,n)? j

Spiel Nr. 2:

Geben Sie Ihren 1. Tip (0 = Ende): 3
Geben Sie Ihren 2. Tip (0 = Ende): 14
Geben Sie Ihren 3. Tip (0 = Ende): 21
Geben Sie Ihren 4. Tip (0 = Ende): 0

Wollen Sie weiterspielen (j,n)? n

Ihr Lottozettel enthaelt folgende Daten:

Spiel Nr. 1:   5  8  17  34  45
Spiel Nr. 2:   3  14  21

Weiteren Zettel ausfuellen (j,n)? n
```

Lösung 7.3: Bruchrechnen

```
MODULE Brueche73 EXPORTS Main;

(* Dieses Programm berechnet alle negativen Potenzen von 5, die sich
   mit maximal 10 Nachkommastellen darstellen lassen.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 26.11.98  Letzte Aenderung: 26.11.98 *)


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


CONST MaxZiffer = 10;  (* Genauigkeit sei maximal 10 Nachkommaziffern *)


TYPE Ziffer = [0 .. 9];
     Bruch  = ARRAY [1 .. MaxZiffer] OF Ziffer;


VAR bruchzahl: Bruch;


(* Prozeduren fuer die Bearbeitung als Array-dargestellter Brueche *)

PROCEDURE TeileBruch(b: Bruch; d: CARDINAL): Bruch=
(* Teilt den Bruch b stellenweise durch die Zahl d und gibt das
   erzielte Ergebnis zurueck *)

VAR rest    : CARDINAL := 0;
    resultat: Bruch;

BEGIN
  FOR i := FIRST(b) TO LAST(b) DO
    rest        := 10 * rest + b[i]; (* Addiere Wert an Stelle i zum Rest *)
    resultat[i] := rest DIV d;       (* Berechne neuen Wert der Stelle i *)
    rest        := rest MOD d;       (* Berechne Rest an der Stelle i *)
  END;
  RETURN resultat;
END TeileBruch;


PROCEDURE DruckeBruch(b: Bruch)=
(* Gibt den Bruch b auf dem Bildschirm aus *)

BEGIN

  (* Gib fuehrende Null und  Dezimalpunkt aus *)
  SIO.PutChar('0'); SIO.PutChar('.');

  (* Gib saemtliche Stellen des Bruchs aus *)
  FOR i := FIRST(b) TO LAST(b) DO
    SIO.PutChar( VAL( b[i] + ORD('0'), CHAR) );
  END;
END DruckeBruch;
```

```
PROCEDURE InitBruch(wert: REAL): Bruch=
(* Initialisiert den Bruch mit dem vorgegebenen REAL-Wert durch wiederholtes
   Multiplizieren mit 10 und Zuweisen des dabei entstehenden ganzzahligen
   Ueberlaufs. *)

VAR resultat: Bruch;

BEGIN
  FOR i := FIRST(resultat) TO LAST(resultat) DO
    wert := wert * 10.0;
    resultat[i] := TRUNC(wert);
    wert := wert - FLOAT(resultat[i], REAL);
  END;
  RETURN resultat;
END InitBruch;


BEGIN
  SIO.PutLine("Die negativen Potenzen von 5 lauten: ");

  (* Initialisiere Bruch mit Startwert 0.2 fuer 5**-1 *)
  bruchzahl := InitBruch(0.2);

  (* Berechne negative Potenzen von 5 mittels Division solange
     genug Stellen vorhanden, d.h. letzte nicht benutzt. *)
  WHILE ( bruchzahl[ LAST(bruchzahl) ] = 0 ) DO
    DruckeBruch(bruchzahl); SIO.Nl();
    bruchzahl := TeileBruch(bruchzahl, 5);
  END;

  (* Drucke letzten Bruch der zum Abbruch fuehrte *)
  DruckeBruch(bruchzahl);
END Brueche73.
```

Die Ausgabe eines Programmlaufs sieht dann wie folgt aus:

```
Die negativen Potenzen von 5 lauten:
0.2000000000
0.0400000000
0.0080000000
0.0016000000
0.0003200000
0.0000640000
0.0000128000
0.0000025600
0.0000005120
0.0000001024
```

Lösungen zu Übung 8

Lösung 8.1: Matrix

a) Mögliche Datenstruktur (siehe auch b)):

```
      TYPE Index = [1..MaxElemente];

           Matrix = ARRAY Index, Index OF CHAR;
```

b)

```
MODULE Matrix81 EXPORTS Main;

(* Dieses Program erzeugt verschiedene Drehungen und Spiegelungen
   einer quadratischen Matrix.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 03.12.98  Letzte Aenderung: 03.12.98 *)


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


CONST MaxElemente = 10; (* Maximale Anzahl Elemente pro Reihe/Spalte der Matrix *)

      (* Konstanten fuer die verschiedenen Operationen *)

      SpiegleVertSymAchse     = 'v';
      SpiegleHoriSymAchse     = 'h';
      SpiegleHauptdiagonale   = 'd';
      SpiegleNebendiagonale   = 'n';
      DreheImUhrzeigersinn    = 'i';
      DreheGegenUhrzeigersinn = 'g';
      Ende                    = 'e';


TYPE Index = [1..MaxElemente];

     Matrix = ARRAY Index, Index OF CHAR;

     Position = RECORD
                  i: Index;
                  j: Index;
                END;

CONST

  (* Initialisiere konstante Zeilen der Matrix mit dem vorgegebenen Muster *)

  Zeile1  = ARRAY Index OF CHAR { '*', '*', '*', '*', '*', '*', '*', '*', '*', '*' };
  Zeile2  = ARRAY Index OF CHAR { ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', '*', ' ' };
  Zeile3  = ARRAY Index OF CHAR { ' ', ' ', ' ', '*', ' ', ' ', ' ', '*', ' ', ' ' };
  Zeile4  = ARRAY Index OF CHAR { ' ', ' ', ' ', ' ', ' ', ' ', '*', ' ', ' ', ' ' };
  Zeile5  = ARRAY Index OF CHAR { ' ', ' ', ' ', ' ', ' ', '*', ' ', ' ', ' ', ' ' };
  Zeile6  = ARRAY Index OF CHAR { ' ', ' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ' };
  Zeile7  = ARRAY Index OF CHAR { ' ', ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ' };
  Zeile8  = ARRAY Index OF CHAR { ' ', ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ' };
  Zeile9  = ARRAY Index OF CHAR { ' ', '*', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ' };
  Zeile10 = ARRAY Index OF CHAR { '*', '*', '*', '*', '*', '*', '*', '*', '*', '*' };
```

```
VAR matrix   := Matrix { Zeile1, Zeile2, Zeile3, Zeile4, Zeile5,
                         Zeile6, Zeile7, Zeile8, Zeile9, Zeile10};
    ergMatrix: Matrix;
    operation: CHAR;
    dummy    : TEXT;


PROCEDURE BestimmePosition(i: Index; j: Index; operation: CHAR): Position=
(* Bestimmt die Position des Elements, das aufgrund des gegebenen
   Parameters operation an die Stelle (i, j) kommt. *)

VAR resultat: Position;

BEGIN
  CASE operation OF
  |  SpiegleVertSymAchse =>
       resultat.i := i;
       resultat.j := LAST(Index) - j + 1;
  |  SpiegleHoriSymAchse =>
       resultat.i := LAST(Index) - i + 1;
       resultat.j := j;
  |  SpiegleHauptdiagonale =>
       resultat.i := j;
       resultat.j := i;
  |  SpiegleNebendiagonale =>
       resultat.i := LAST(Index) - j + 1;
       resultat.j := LAST(Index) - i + 1;
  |  DreheImUhrzeigersinn =>
       resultat.i := LAST(Index) - j + 1;
       resultat.j := i;
  |  DreheGegenUhrzeigersinn =>
       resultat.i := j;
       resultat.j := LAST(Index) - i + 1;
  ELSE
    (* Undefinierte Operation, gib Position unveraendert zurueck *)
    resultat.i := i;
    resultat.j := j;
  END;

  RETURN resultat;
END BestimmePosition;


PROCEDURE BestimmeMatrix(READONLY matrix: Matrix; VAR ergMatrix: Matrix;
                         operation: CHAR)=
(* Bestimmt die aufgrund des gegebenen Parameters operation umgespeicherte
   Ergebnismatrix ergMatrix aus der gegebenen Ursprungsmatrix matrix *)

VAR urPosition: Position;

BEGIN
  FOR i := 1 TO LAST(Index) DO
    FOR j := 1 TO LAST(Index) DO
      urPosition := BestimmePosition(i, j, operation);
      ergMatrix[i, j] := matrix[urPosition.i, urPosition.j];
    END;
  END;
END BestimmeMatrix;
```

```
PROCEDURE DruckeMatrix(READONLY matrix: Matrix)=
(* Gibt die gegebene Matrix reihenweise aus *)
BEGIN
  FOR i := 1 TO LAST(Index) DO
    FOR j := 1 TO LAST(Index) DO
      SIO.PutChar(matrix[i, j]);
    END;
    SIO.Nl();
  END;
END DruckeMatrix;


BEGIN
  REPEAT
    SIO.PutLine("Es gibt folgende Operationen zur Manipulation" &
                " der Matrix: v, h, d, n, i, g. e = Ende!");
    SIO.PutText("Welche davon soll ausgefuehrt werden? ");

    (* Lies Eingabe und RETURN *)
    operation := SIO.GetChar();
    dummy := SIO.GetLine();     (* Entfernt RETURN aus der Eingabe *)

    IF (operation # Ende) THEN
      BestimmeMatrix(matrix, ergMatrix, operation);
      SIO.Nl();
      DruckeMatrix(ergMatrix);
      SIO.Nl();
    END;
  UNTIL operation = Ende;
END Matrix81.
```

Ein möglicher Programmlauf (**mit Benutzereingaben**) produziert folgende Ausgaben:

```
Es gibt folgende Operationen zur Manipulation der Matrix: v, h, d, n, i, g. e = Ende!
Welche davon soll ausgefuehrt werden? v
**********
 *
  *   *
   *
    *
     *
      *
       *
        *
**********

Es gibt folgende Operationen zur Manipulation der Matrix: v, h, d, n, i, g. e = Ende!
Welche davon soll ausgefuehrt werden? n
*        *
*       **
*      * *
*     *  *
*    *   *
*   *    *
*  *   * *
* *      *
**       *
*        *

Es gibt folgende Operationen zur Manipulation der Matrix: v, h, d, n, i, g. e = Ende!
Welche davon soll ausgefuehrt werden? e
```

Lösung 8.2: Schach

a) Möglicher Datentyp (mit Teil-Datentypen) für ein Schachbrett:

```
     Farbe = {Schwarz, Weiss};

     Figur = {Bauer, Springer, Laeufer, Turm, Dame, Koenig};

     Belegung = RECORD
                  belegt: BOOLEAN;
                  farbe : Farbe;
                  figur : Figur;
                END;

     XIndex = [1..8];
     YIndex = [1..8];

     Schachbrett = ARRAY XIndex, YIndex OF Belegung;
```

b) Siehe Prozeduren `ErmittleZuegeLaeufer` und `ErmittleZuegeSpringer` in Teil c).

c)

```
MODULE Schach82 EXPORTS Main;

(* Dieses Program ermittelt moegliche Zuege und schlagbare Figuren in einem
   Schachspiel.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 02.12.98  Letzte Aenderung: 02.12.98 *)


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


CONST MaxZuege = 28; (* Maximale Anzahl moeglicher Zuege *)


TYPE
     (* Datentypen fuer die Stellung auf einem Schachfeld *)

     Farbe = {Schwarz, Weiss};

     Figur = {Bauer, Springer, Laeufer, Turm, Dame, Koenig};

     Belegung = RECORD
                  belegt: BOOLEAN;
                  farbe : Farbe;
                  figur : Figur;
                END;

     XIndex = [1..8];
     YIndex = [1..8];

     Schachbrett = ARRAY XIndex, YIndex OF Belegung;

     (* Datentypen fuer Menge der moeglichen Zuege einer Figur *)

     ZugZahl = [0..MaxZuege];

     Position = RECORD
                  x: XIndex;
                  y: YIndex;
                END;
```

```
    Zuege = RECORD
              anzahl: CARDINAL;
              position: ARRAY ZugZahl OF Position;
            END;

CONST
      (* Definiere Konstanten fuer bei der Initialisierung benoetigte Belegungen *)

      LeeresFeld        = Belegung {belegt := FALSE, farbe := Farbe.Weiss,
                                    figur := Figur.Bauer};
      WeisserBauer      = Belegung {belegt := TRUE, farbe := Farbe.Weiss,
                                    figur := Figur.Bauer};
      SchwarzerBauer    = Belegung {belegt := TRUE, farbe := Farbe.Schwarz,
                                    figur := Figur.Bauer};
      SchwarzerSpringer = Belegung {belegt := TRUE, farbe := Farbe.Schwarz,
                                    figur := Figur.Springer};
      WeisserLaeufer    = Belegung {belegt := TRUE, farbe := Farbe.Weiss,
                                    figur := Figur.Laeufer};
      WeisserKoenig     = Belegung {belegt := TRUE, farbe := Farbe.Weiss,
                                    figur := Figur.Koenig};
      SchwarzerKoenig   = Belegung {belegt := TRUE, farbe := Farbe.Schwarz,
                                    figur := Figur.Koenig};


VAR schachbrett := Schachbrett { ARRAY YIndex OF Belegung {LeeresFeld,..},..};
    zuege: Zuege;
    feld : Position;
PROCEDURE InitZuege(VAR zuege: Zuege)=
(* Initialisiere Datenstruktur Zuege mit 0 Zuegen *)
BEGIN
  zuege.anzahl := 0;
END InitZuege;


PROCEDURE SetzeZug(x: XIndex; y: YIndex; VAR zuege: Zuege)=
(* Setze neuen Zug im Parameter zuege auf Feld mit den Koordinaten x, y *)
BEGIN
  zuege.anzahl := zuege.anzahl + 1;
  zuege.position[zuege.anzahl].x := x;
  zuege.position[zuege.anzahl].y := y;
END SetzeZug;


PROCEDURE ZugGueltig(READONLY schachbrett: Schachbrett;
                     x, y: INTEGER; farbe: Farbe): BOOLEAN=
(* Prueft, ob ein Zug auf die Position (x, y) gueltig ist *)
VAR resultat: BOOLEAN;

BEGIN
  IF (x < FIRST(XIndex)) OR (x > LAST(XIndex)) OR
     (y < FIRST(YIndex)) OR (y > LAST(YIndex))
  THEN
    (* Zug nicht mehr im Bereich des Schachbretts *)
    resultat := FALSE;
  ELSIF (schachbrett[x, y].belegt) AND
        (schachbrett[x, y].farbe = farbe)
  THEN
    (* Feld bereits durch eigene Figur belegt *)
    resultat := FALSE;
  ELSE
    (* Zug ok *)
    resultat := TRUE;
  END;

  RETURN resultat;
END ZugGueltig;
```

```
PROCEDURE ErmittleZuege(READONLY schachbrett: Schachbrett;
                        feld: Position; VAR zuege: Zuege)=
(* Ermittelt alle Zuege, die von der Position feld aus moeglich sind *)
BEGIN
  InitZuege(zuege); (* Initialisiere Variable moeglicher Zuege mit 0 Zuegen *)

  (* Pruefe, ob tatsaechlich Figur an der Position steht *)
  IF schachbrett[feld.x, feld.y].belegt THEN

    (* Ermittle Zuege abhaengig von der vorliegenden Figur *)
    CASE schachbrett[feld.x, feld.y].figur OF
    |  Figur.Springer => ErmittleZuegeSpringer(schachbrett, feld, zuege);
    |  Figur.Laeufer  => ErmittleZuegeLaeufer (schachbrett, feld, zuege);

       (* Hier fehlen noch ein paar... *)

    END;

  END; (* IF *)
END ErmittleZuege;


PROCEDURE ErmittleZuegeSpringer(READONLY schachbrett: Schachbrett;
                                feld: Position; VAR zuege: Zuege)=
(* Prueft alle durch einen Springer von Position feld aus
   erreichbaren Felder auf moegliche, gueltige Zuege *)

VAR y    : INTEGER;
    farbe: Farbe;

BEGIN
  (* Bestimme Farbe der Figur an der Position feld *)
  farbe := schachbrett[feld.x, feld.y].farbe;

  (* Pruefe alle durch einen Springer erreichbaren Felder *)
  FOR x := -2 TO 2 DO
    IF (x # 0) THEN (* x = 0 mit Springer nicht moeglich *)

      y := 3 - ABS(x); (* Springerzug geht stets ueber 3 Felder *)

      (* Pruefe symmetrische Zuege fuer y und -y *)
      IF ZugGueltig(schachbrett, feld.x+x, feld.y+y, farbe) THEN
        SetzeZug(feld.x+x, feld.y+y, zuege);
      END;
      IF ZugGueltig(schachbrett, feld.x+x, feld.y-y, farbe) THEN
        SetzeZug(feld.x+x, feld.y-y, zuege);
      END;
    END;
  END; (* FOR x *)
END ErmittleZuegeSpringer;
```

```
PROCEDURE ErmittleZuegeLaeufer(READONLY schachbrett: Schachbrett;
                               feld: Position; VAR zuege: Zuege)=
(* Prueft alle durch einen Laeufer von der Position feld aus
   erreichbaren Felder auf moegliche, gueltige Zuege *)

VAR x, y      : INTEGER;
    farbe     : Farbe;

BEGIN
  (* Bestimme Farbe der Figur an der Position feld *)
  farbe := schachbrett[feld.x, feld.y].farbe;

  (* Ueberpruefe die Diagonalen mit Steigungen: (-1, -1), (-1, 1),
                                                (1, -1),  (1, 1) *)
  FOR xInk := -1 TO 1 DO
    FOR yInk := -1 TO 1 DO

      IF (xInk # 0) AND (yInk # 0) THEN (* x = 0 oder y = 0 nicht moeglich *)

        (* Pruefe Zuege auf der Diagonalen, bis nicht mehr gueltig (Brett zu Ende,
           eigene Figur im Weg) oder Feld durch gegnerische Figur belegt (kann
           nicht uebersprungen werden) *)
        x := xInk; y := yInk;
        WHILE ZugGueltig(schachbrett, feld.x+x, feld.y+y, farbe) AND
              NOT schachbrett[feld.x+x, feld.y+y].belegt
        DO
          SetzeZug(feld.x+x, feld.y+y, zuege);
          x := x + xInk;
          y := y + yInk;
        END;

        IF ZugGueltig(schachbrett, feld.x+x, feld.y+y, farbe) AND
           schachbrett[feld.x+x, feld.y+y].belegt
        THEN
           (* Gegnerische Figur kann geschlagen werden !! *)
           SetzeZug(feld.x+x, feld.y+y, zuege);
        END;
      END; (* IF *)

    END; (* FOR y *)
  END; (* FOR x *)
END ErmittleZuegeLaeufer;


PROCEDURE GibFigurName(figur: Figur): TEXT=
(* Gibt den Namen einer Figur als Text zurueck *)

VAR resultat: TEXT;

BEGIN
  CASE figur OF
  | Figur.Bauer    => resultat := "Bauer";
  | Figur.Springer => resultat := "Springer";
  | Figur.Laeufer  => resultat := "Laeufer";
  | Figur.Turm     => resultat := "Turm";
  | Figur.Dame     => resultat := "Dame";
  | Figur.Koenig   => resultat := "Koenig";
  END;

  RETURN resultat;
END GibFigurName;
```

```
PROCEDURE DruckeZuege(READONLY schachbrett: Schachbrett; READONLY zuege: Zuege)=
(* Druckt alle in zuege enthaltenen Positionen und gibt eventuell an,
   welche gegnerischen Figuren dabei geschlagen werden *)
BEGIN
  SIO.PutLine("Moegliche Zuege: "); SIO.Nl();

  FOR i := 1 TO zuege.anzahl DO
    WITH x = zuege.position[i].x,
         y = zuege.position[i].y
    DO
      (* Gib mit Zug erreichbare Position aus *)
      SIO.PutText("Zug "); SIO.PutInt(i); SIO.PutText(" : ");
      SIO.PutInt(x); SIO.PutText(", "); SIO.PutInt(y);

      (* Pruefe, ob Figur an der Stelle geschlagen wird *)
      IF schachbrett[x, y].belegt THEN
        SIO.PutLine(" schlaegt " & GibFigurName(schachbrett[x, y].figur));
      ELSE
        SIO.Nl();
      END;
    END; (* WITH *)
  END;
END DruckeZuege;


PROCEDURE InitSchachbrett(VAR sb: Schachbrett)=
(* Belegt das Schachbrett sb mit der vorgegebenen Teststellung *)
BEGIN
  sb[4, 5] := SchwarzerSpringer;
  sb[3, 7] := WeisserLaeufer;

  sb[3, 3] := WeisserLaeufer;
  sb[4, 3] := WeisserBauer;
  sb[2, 4] := SchwarzerBauer;
  sb[1, 5] := SchwarzerBauer;
  sb[6, 4] := WeisserBauer;
  sb[5, 7] := SchwarzerBauer;
  sb[4, 8] := SchwarzerSpringer;
  sb[7, 8] := WeisserKoenig;
  sb[8, 1] := SchwarzerKoenig;
END InitSchachbrett;


BEGIN
  (* Initialisiere Schachbrett mit der Teststellung *)
  InitSchachbrett(schachbrett);

  (* Ermittle Zuege fuer schwarzen Springer an Position 4, 5 *)
  feld.x := 4; feld.y := 5;
  ErmittleZuege(schachbrett, feld, zuege);
  DruckeZuege(schachbrett, zuege);
  SIO.Nl();

  (* Ermittle Zuege fuer weissen Laeufer an Position 3, 7 *)
  feld.x := 3; feld.y := 7;
  ErmittleZuege(schachbrett, feld, zuege);
  DruckeZuege(schachbrett, zuege);
END Schach82.
```

Die Ausgabe eines Programmlaufs sieht dann folgendermaßen aus:

```
Moegliche Zuege:

Zug 1 : 2, 6
Zug 2 : 3, 7 schlaegt Laeufer
Zug 3 : 3, 3 schlaegt Laeufer
Zug 4 : 5, 3
Zug 5 : 6, 6
Zug 6 : 6, 4 schlaegt Bauer

Moegliche Zuege:

Zug 1 : 2, 6
Zug 2 : 1, 5 schlaegt Bauer
Zug 3 : 2, 8
Zug 4 : 4, 6
Zug 5 : 5, 5
Zug 6 : 4, 8 schlaegt Springer
```

Lösungen zu Übung 9

Lösung 9.1: Scanner

```
MODULE Scanner91 EXPORTS Main;

(* Dieses Programm implementiert einen simplen Scanner fuer Benutzer-Eingaben.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 10.12.98  Letzte Aenderung: 18.12.98 *)

IMPORT SIO;  (* Importiere Ein-/Ausgabeoperationen *)
IMPORT Text; (* Importiere Operationen zur Textmanipulation *)

CONST Buchstaben       = SET OF CHAR {'a' .. 'z', 'A' .. 'Z'};
      Leerzeichen      = SET OF CHAR {' ', '\t', '\r', '\n'};
      Ziffern          = SET OF CHAR {'0' .. '9'};
      Zusammengesetzte = SET OF CHAR {'>', '<', ':'};

TYPE SymbolArt = {Bezeichner, Zahl, Plus, Minus, Mal, Durch, Kleiner,
                  KleinerGleich, Groesser, GroesserGleich, Gleich, Ungleich,
                  ErgibtSich, Doppelpunkt, Semikolon, Unbekannt};

     Symbol    = RECORD
                   art    : SymbolArt;
                   text   : TEXT;     (* u.a. Wert des Bezeichners *)
                   wert   : INTEGER;  (* Wert der Zahl *)
                 END;

VAR zeichen: CHAR;
    symbol : Symbol;


PROCEDURE LeseSymbol(VAR zeichen: CHAR): Symbol=
(* Liest ein Symbol (Bezeichner, Zahl, ...) von der Standardeingabe.
   Der Parameter zeichen enthaelt das zuletzt gelesene, jedoch nicht
   verwertete Zeichen. *)

VAR resultat: Symbol;

BEGIN
  (* Ueberspringe Leerzeichen *)
  WHILE zeichen IN Leerzeichen DO
    zeichen := SIO.GetChar();
  END;

  (* Anstehendes Zeichen bestimmt, welches Symbol vorliegt *)
  IF zeichen IN Buchstaben THEN
    (* Lese den Bezeichner als Symbol ein *)
    resultat := LeseBezeichner(zeichen);
  ELSIF zeichen IN Ziffern THEN
    (* Lese eine Zahl als Symbol ein *)
    resultat := LeseZahl(zeichen);
  ELSIF zeichen IN Zusammengesetzte THEN
    (* Versuche ein zusammengesetztes Symbol, z.B. <= , zu lesen *)
    resultat := LeseZusammengesetztesSymbol(zeichen);
  ELSE
```

```
    (* Bestimme das Symbol fuer ein einzelnes Zeichen *)
    resultat := BestimmeEinfachesSymbol(zeichen);
    zeichen := SIO.GetChar();
  END;

  RETURN resultat;
END LeseSymbol;

PROCEDURE LeseBezeichner(VAR zeichen: CHAR): Symbol=
(* Liest einen Bezeichner von der Standardeingabe und
   erzeugt das entsprechende Symbol. Der Parameter zeichen
   enthaelt das zuletzt gelesene Zeichen, hier den Anfang des
   Bezeichners. *)

VAR resultat: Symbol;

BEGIN
  resultat.art := SymbolArt.Bezeichner;

  (* Lege den Bezeichner in der Komponente text ab *)
  resultat.text := "Bezeichner: ";
  REPEAT
    resultat.text := resultat.text & Text.FromChar(zeichen);
    zeichen := SIO.GetChar();
  UNTIL NOT ((zeichen IN Buchstaben) OR (zeichen IN Ziffern));

  RETURN resultat;
END LeseBezeichner;


PROCEDURE LeseZahl(VAR zeichen: CHAR): Symbol=
(* Liest eine Zahl von der Standardeingabe und erzeugt das
   entsprechende Symbol. Der Parameter zeichen enthaelt das
   zuletzt gelesene Zeichen, hier die erste Ziffer der Zahl. *)

VAR resultat: Symbol;

BEGIN
  resultat.art := SymbolArt.Zahl;

  (* Speichere den Wert der Zahl in der Komponente wert *)
  resultat.wert := 0;
  REPEAT
    resultat.wert := resultat.wert * 10 + (ORD(zeichen) - ORD('0'));
    zeichen := SIO.GetChar();
  UNTIL NOT (zeichen IN Ziffern);

  RETURN resultat;
END LeseZahl;
```

```
PROCEDURE LeseZusammengesetztesSymbol(VAR zeichen: CHAR): Symbol=
(* Liest zwei zusammengesetzte Zeichen (<=, >=, :=) von der
   Standardeingabe, wenn moeglich, und erzeugt das entsprechende Symbol.
   Ist dies nicht moeglich wird fuer das gegebene zuletzt gelesene Zeichen
   das entsprechende Symbol erzeugt (<, >, :). *)

VAR resultat     : Symbol;
    erstesZeichen: CHAR;

BEGIN
  (* Speichere gelesenes Zeichen und lies weiteres Zeichen fuer Auswertung *)
  erstesZeichen := zeichen;
  zeichen := SIO.GetChar();

  IF (zeichen = '=') THEN

    (* Zusammengesetztes Symbol erkannt *)
    CASE erstesZeichen OF
    |  '<' => resultat.art := SymbolArt.KleinerGleich;
              resultat.text := "KleinerGleich";
    |  '>' => resultat.art := SymbolArt.GroesserGleich;
              resultat.text := "GroesserGleich";
    |  ':' => resultat.art := SymbolArt.ErgibtSich;
              resultat.text := "ErgibtSich";
       (* Theoretisch kein ELSE-Zweig moeglich! *)
    END;



    (* Lies neues Zeichen fuer weiteren Verlauf *)
    zeichen := SIO.GetChar();
  ELSE
    (* Kein zusammengesetztes Symbol *)
    resultat := BestimmeEinfachesSymbol(erstesZeichen);
  END;

  RETURN resultat;
END LeseZusammengesetztesSymbol;


PROCEDURE BestimmeEinfachesSymbol(zeichen: CHAR): Symbol=
(* Bestimmt das Symbol zu einem einzelnen Zeichen oder
   das Symbol Unbekannt, wenn das nicht moeglich ist. *)

VAR resultat: Symbol;

BEGIN
  CASE zeichen OF
  | '+'  => resultat.art := SymbolArt.Plus;
            resultat.text := "Plus";
  | '-'  => resultat.art := SymbolArt.Minus;
            resultat.text := "Minus";
  | '*'  => resultat.art := SymbolArt.Mal;
            resultat.text := "Mal";
  | '/'  => resultat.art := SymbolArt.Durch;
            resultat.text := "Durch";
  | ':'  => resultat.art := SymbolArt.Doppelpunkt;
            resultat.text := "Doppelpunkt";
  | '<'  => resultat.art := SymbolArt.Kleiner;
            resultat.text := "Kleiner";
  | '>'  => resultat.art := SymbolArt.Groesser;
            resultat.text := "Groesser";
  | '='  => resultat.art := SymbolArt.Gleich;
            resultat.text := "Gleich";
```

```
  | '#'  => resultat.art := SymbolArt.Ungleich;
          resultat.text := "Ungleich";
  | ';'  => resultat.art := SymbolArt.Semikolon;
          resultat.text := "Semikolon";
  ELSE
    resultat.art := SymbolArt.Unbekannt;
    resultat.text := "Unbekannt: " & Text.FromChar(zeichen);
  END;

  RETURN resultat;
END BestimmeEinfachesSymbol;


BEGIN
  (* Lese Symbole in Standardeingabe, bis unbekanntes Symbol entdeckt wird. *)
  zeichen := SIO.GetChar();
  REPEAT
    symbol := LeseSymbol(zeichen);
    IF (symbol.art = SymbolArt.Zahl) THEN

      (* Gib den Zahlenwert aus *)
      SIO.PutText("Zahl: ");
      SIO.PutInt(symbol.wert);
      SIO.Nl();

    ELSE

      (* Gib Bezeichner bzw. Beschreibung aus *)
      SIO.PutLine(symbol.text);

    END;
  UNTIL (symbol.art = SymbolArt.Unbekannt);
END Scanner91.
```

Ein Dialog mit dem Scanner sieht dann zum Beispiel (**mit Benutzereingaben**) so aus:

```
x:= 5 - 7*wert/zahl
Bezeichner: x
ErgibtSich
Zahl: 5
Minus
Zahl: 7
Mal
Bezeichner: wert
Durch
Bezeichner: zahl
x := a<==>>=wert
Bezeichner: x
ErgibtSich
Bezeichner: a
KleinerGleich
Gleich
Groesser
GroesserGleich
Bezeichner: wert
ende ?
Bezeichner: ende
Unbekannt: ?
```

Lösung 9.2: Tennisrangliste

```
MODULE Tennis92 EXPORTS Main;

(* Dieses Programm dient der Verwaltung einer Tennisrangliste.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 09.12.98  Letzte Aenderung: 10.12.98 *)


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


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

     Spieler = RECORD
                 name       : Name;
                 mitgliedsNr: CARDINAL;
               END;

     RanglisteRef = REF RanglistenElement;

     RanglistenElement = RECORD
                           spieler  : Spieler;
                           naechster: RanglisteRef;
                         END;

VAR rangliste    : RanglisteRef; (* Anker der Club-Rangliste *)
    spieler,
    gewinner,
    verlierer    : Spieler;
    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 SucheSpielerRL(liste: RanglisteRef; spieler: Spieler): RanglisteRef=
(* Sucht den gegebenen Spieler in der Rangliste liste und gibt einen Zeiger
   auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der Spieler 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 Spieler der gesuchte ist *)
     IF (Text.Equal(resRLRef^.spieler.name.vorname, spieler.name.vorname)) AND
        (Text.Equal(resRLRef^.spieler.name.nachname, spieler.name.nachname)) AND
        (resRLRef^.spieler.mitgliedsNr = spieler.mitgliedsNr)
     THEN
       gefunden := TRUE;
     ELSE
       resRLRef := resRLRef^.naechster;
     END;

   END; (* WHILE *)

   RETURN resRLRef;
END SucheSpielerRL;


PROCEDURE SucheVorgaengerRL(liste: RanglisteRef; spieler: Spieler):
RanglisteRef=
(* Sucht den Vorgaenger des gegebenen Spielers in der Rangliste liste und gibt
   einen Zeiger auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der Spieler
   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 Spieler der gesuchte ist *)
     IF (Text.Equal(aktRLRef^.spieler.name.vorname, spieler.name.vorname)) AND
        (Text.Equal(aktRLRef^.spieler.name.nachname, spieler.name.nachname)) AND
        (aktRLRef^.spieler.mitgliedsNr = spieler.mitgliedsNr)
     THEN
       gefunden := TRUE;
     ELSE
       vorRLRef := aktRLRef;
       aktRLRef := aktRLRef^.naechster;
     END;

   END; (* WHILE *)
```

```
   (* Zeiger auf den Vorgaenger ist NIL, wenn Spieler 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(" *** Tennis-Rangliste *** ");
  SIO.Nl();

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

    (* Gib Rang und Daten des Spielers aus *)
    SIO.PutInt(rang); SIO.PutText(". ");
    SIO.PutText(ranglisteRef^.spieler.name.nachname & " ");
    SIO.PutText(ranglisteRef^.spieler.name.vorname & "  ");
    SIO.PutText("Mitgl.-Nr. ");
    SIO.PutInt(ranglisteRef^.spieler.mitgliedsNr);
    SIO.Nl();

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

  END; (* WHILE *)
END DruckeRL;


PROCEDURE LoescheSpielerRL(VAR liste: RanglisteRef; spieler: Spieler)=
(* Loescht den gegebenen Spieler aus der Rangliste liste, falls vorhanden. *)

VAR ranglisteRef,
    vorgaengerRef: RanglisteRef;

BEGIN
  (* Suche den Spieler in der Liste *)
  ranglisteRef := SucheSpielerRL(liste, spieler);

  IF (ranglisteRef # NIL) THEN

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

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

```
PROCEDURE EinfuegenSpielerRL(VAR liste: RanglisteRef; spieler: Spieler)=
(* Fuegt den gegebenen Spieler am Ende der Rangliste liste ein *)

VAR ranglisteRef, aktRLRef: RanglisteRef;

BEGIN
  (* Pruefe, ob Spieler nicht schon in Rangliste enthalten *)
  ranglisteRef := SucheSpielerRL(liste, spieler);

  IF (ranglisteRef = NIL) THEN

    (* Erzeuge neues Ranglisten-Element fuer den Spieler *)
    ranglisteRef := NEW(RanglisteRef);
    ranglisteRef^.spieler := spieler;
    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 Spieler ans Ende an *)
      aktRLRef^.naechster := ranglisteRef;
    END;
  ELSE
    SIO.PutLine("FEHLER: Spieler ist bereits in der Rangliste!");
  END;
END EinfuegenSpielerRL;


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: Spieler)=
(* Aendert die Rangfolge in der Rangliste entsprechend dem gegebenen
   Spielergebnis (Gewinner, Verlierer) *)

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

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

  IF (gewinnerRef = NIL) OR (verliererRef = NIL) THEN
    SIO.PutLine("FEHLER: Wenigstens einer der Spieler 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 Gewinner vor Verlierer in Rangliste ein *)

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

      (* Haenge Gewinner ein bei Vorgaenger des Verlierers *)
      IF ( vorVerliererRef = NIL) THEN
        (* Kein Vorgaenger, da ganz am Anfang der Liste *)
        liste := gewinnerRef;
      ELSE
        vorVerliererRef^.naechster := gewinnerRef;
      END;

      (* Haenge Nachfolger des Gewinners bei seinem Vorgaenger ein.
         BEACHTE: Wenigstens der Verlierer kommt hier vor dem Gewinner in der
         Liste, so dass die Referenz vorGewinnerRef nicht NIL sein kann. *)
      vorGewinnerRef^.naechster := gewinnerRef^.naechster;

      (* Haenge Verlierer als Nachfolger des Gewinners ein *)
      gewinnerRef^.naechster := verliererRef;
    END;

  END;
END AktualisiereRL;
```

```
PROCEDURE LeseSpielerEin(nachricht: TEXT): Spieler=
(* Fragt den Benutzer nach den Daten eines Spielers und gibt
   anschliessend einen entsprechenden Spieler zurueck *)

VAR resSpieler     : Spieler;
    dummy          : TEXT;

BEGIN
  SIO.PutLine(nachricht);
  SIO.PutText("Nachname   : ");
  resSpieler.name.nachname := SIO.GetLine();
  SIO.PutText("Vorname    : ");
  resSpieler.name.vorname := SIO.GetLine();
  SIO.PutText("Mitgliedsnr: ");
  resSpieler.mitgliedsNr := SIO.GetInt();

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

  RETURN resSpieler;
END LeseSpielerEin;

PROCEDURE DruckeMenue()=
(* Hilfsprozedur, die das Befehlsmenue ausgibt *)
BEGIN
  SIO.Nl();
  SIO.PutLine("Z : Zeige Rangliste an");
  SIO.PutLine("F : Fuege einen Spieler ein");
  SIO.PutLine("L : Loesche einen Spieler");
  SIO.PutLine("G : Spiel 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' => spieler := LeseSpielerEin("Spieler Einfuegen");
                   SIO.Nl();
                   EinfuegenSpielerRL(rangliste, spieler);
    |  'L', 'l' => spieler := LeseSpielerEin("Spieler Loeschen");
                   SIO.Nl();
                   LoescheSpielerRL(rangliste, spieler);
    |  'G', 'g' => gewinner := LeseSpielerEin("Sieger des Spiels");
                   SIO.Nl();
                   verlierer := LeseSpielerEin("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 Tennis92.
```

Ein möglicher Dialog mit dem Programm sieht dann (**mit Benutzereingaben**) folgendermaßen aus:

```
Z : Zeige Rangliste an
F : Fuege einen Spieler ein
L : Loesche einen Spieler
G : Spiel gemacht
E : Exit

Auswahl: f

Spieler Einfuegen
Nachname   : Nase
Vorname    : As
Mitgliedsnr: 11

Z : Zeige Rangliste an
...
E : Exit

Auswahl: f

Spieler Einfuegen
Nachname   : Pappnase
Vorname    : Flip
Mitgliedsnr: 3

Z : Zeige Rangliste an
...
E : Exit

Auswahl: f

Spieler Einfuegen
Nachname   : Oberas
Vorname    : D.
Mitgliedsnr: 7

Z : Zeige Rangliste an ...
E : Exit

Auswahl: g

Sieger des Spiels
Nachname   : Oberas
Vorname    : D.
Mitgliedsnr: 7

Verlierer des Spiels
Nachname   : Nase
Vorname    : As
Mitgliedsnr: 11

Z : Zeige Rangliste an
...
E : Exit

Auswahl: z

 *** Tennis-Rangliste ***

1. Oberas D.  Mitgl.-Nr. 7
2. Nase As  Mitgl.-Nr. 11
3. Pappnase Flip  Mitgl.-Nr. 3
```

```
Z : Zeige Rangliste an
...
E : Exit

Auswahl: e
```

Lösungen zu Übung 10

Lösung 10.1: Ruprechts Lagerverwaltung

```
MODULE Ruprecht101 EXPORTS Main;

(* Dieses Programm realisiert eine einfache Lagerverwaltung
   fuer Knecht Ruprecht.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 17.12.98  Letzte Aenderung: 22.12.98 *)


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


CONST MindestAnzahl = 5;  (* der Geschenke pro Art *)


TYPE ArtRef      = REF Art;

     Art         = RECORD
                     art      : TEXT;
                     anzahl   : CARDINAL;    (* Geschenke einer Art *)
                     vorrat   : GeschenkRef;
                     naechster: ArtRef;
                   END;

     GeschenkRef = REF Geschenk;

     Geschenk    = RECORD
                     beschreibung: TEXT;
                     naechstes   : GeschenkRef;
                   END;


VAR lager  : ArtRef;  (* Anker der Datenstruktur fuer das Geschenklager *)
    auswahl: CHAR;
    dummy  : CHAR;

(* Hilfsprozeduren *)

PROCEDURE ErfrageArt(): TEXT=
(* Fragt den Benutzer nach der Art eines Geschenks *)
BEGIN
  SIO.PutText("Geschenkart: ");
  RETURN SIO.GetLine();
END ErfrageArt;


PROCEDURE ErfrageBeschreibung(): TEXT=
(* Fragt den Benutzer nach der Beschreibung eines Geschenks *)
BEGIN
  SIO.PutText("Beschreibung des Geschenks: ");
  RETURN SIO.GetLine();
END ErfrageBeschreibung;
```

```
PROCEDURE ErfrageOperation(): CHAR=
(* Druckt alle erlaubten Befehle aus und fragt
   Benutzer nach seiner Auswahl *)
VAR auswahl: CHAR;

BEGIN
  SIO.Nl();
  SIO.PutLine("Neue Geschenkart aufnehmen: a");
  SIO.PutLine("Neues Geschenk einlagern  : n");
  SIO.PutLine("Geschenk auslagern        : l");
  SIO.PutLine("Geschenkstatistik erzeugen: s");
  SIO.PutLine("Ende                      : e");
  SIO.Nl();

  SIO.PutText("Auswahl: ");
  auswahl := SIO.GetChar();
  SIO.Nl();

  (* Loesche RETURN aus der Eingabe *)
  WHILE SIO.Available() DO
    dummy := SIO.GetChar();
  END;

  RETURN auswahl;
END ErfrageOperation;


(* Prozeduren zum Datentyp ArtRef *)

PROCEDURE InitLager(VAR lager: ArtRef)=
(* Initialisiere Ruprechts leeres Lager *)
BEGIN
  lager := NIL;
END InitLager;


PROCEDURE ErzeugeArt(artName: TEXT): ArtRef=
(* Erzeugt eine neue Geschenkart in der Halde fuer
   die gegebene Geschenkart und gibt den Zeiger darauf
   zurueck *)

VAR resultatRef: ArtRef;

BEGIN
  resultatRef            := NEW(ArtRef);
  resultatRef^.art       := artName;
  resultatRef^.anzahl    := 0;
  resultatRef^.vorrat    := NIL;
  resultatRef^.naechster := NIL;

  RETURN resultatRef;
END ErzeugeArt;
```

```
PROCEDURE AufnehmenArt(VAR lager: ArtRef; neuArtRef: ArtRef)=
(* Fuegt die neue Geschenkart an das Ende der Liste an *)

VAR aktArtRef: ArtRef;

BEGIN
  IF (lager = NIL) THEN
    (* Lager noch leer *)
    lager := neuArtRef;
  ELSE
    (* Suche Ende der Geschenkartliste *)
    aktArtRef := lager;
    WHILE (aktArtRef^.naechster # NIL) DO
      aktArtRef := aktArtRef^.naechster;
    END;
    aktArtRef^.naechster := neuArtRef;
  END;
END AufnehmenArt;


PROCEDURE AufnehmenArtDialog(VAR lager: ArtRef)=
(* Fragt den Benutzer nach einer neu aufzunehmenden Geschenkart
   und nimmt diese, wenn noch nicht vorhanden, ins Lager auf *)

VAR artName  : TEXT;
    neuArtRef: ArtRef;

BEGIN
  (* Frage Benutzer nach neuer Geschenkart *)
  SIO.PutLine("Aufnehmen einer neuen Geschenkart: ");
  artName := ErfrageArt();

  IF NOT ExistiertArt(lager, artName) THEN

    (* Lege neue Geschenkart an, da noch nicht enthalten *)
    neuArtRef := ErzeugeArt(artName);
    AufnehmenArt(lager, neuArtRef);

  ELSE
    SIO.PutLine("Geschenkart bereits enthalten!");
  END;
END AufnehmenArtDialog;


PROCEDURE SucheArt(lager: ArtRef; artName: TEXT): ArtRef=
(* Suche nach der Geschenkart artName in der Liste und gib einen Zeiger
   darauf zurueck oder NIL, wenn nicht enthalten *)

VAR resultatRef: ArtRef;

BEGIN
  resultatRef := lager;
  WHILE (resultatRef # NIL) AND
        (Text.Compare(resultatRef^.art, artName) # 0)
  DO
    resultatRef := resultatRef^.naechster;
  END;

  RETURN resultatRef;
END SucheArt;
```

```
PROCEDURE ExistiertArt(lager: ArtRef; artName: TEXT): BOOLEAN=
(* Stellt fest, ob eine Geschenkart bereits in der Liste existiert
   oder nicht *)
BEGIN
  RETURN (SucheArt(lager, artName) # NIL);
END ExistiertArt;


PROCEDURE GeschenkAnzahlOK(aktArtRef: ArtRef): BOOLEAN=
(* Pruefe, ob vorgegebene Mindestanzahl Geschenke fuer diese Art vorhanden *)
BEGIN
  RETURN NOT (aktArtRef^.anzahl < MindestAnzahl);
END GeschenkAnzahlOK;


PROCEDURE DruckeLagerinhalt(lager: ArtRef)=
(* Druckt alle Geschenke im Lager aus *)

VAR aktArtRef: ArtRef;

BEGIN
  SIO.PutLine("Aktueller Lagerinhalt: ");
  SIO.Nl();

  (* Drucke alle Geschenkarten mit den zugehoerigen Geschenken aus *)
  aktArtRef := lager;
  WHILE (aktArtRef # NIL) DO
    SIO.PutLine(aktArtRef^.art & ": ");
    DruckeGeschenke(aktArtRef^.vorrat);
    SIO.Nl();
    aktArtRef := aktArtRef^.naechster;
  END;

END DruckeLagerinhalt;

(* Prozeduren zum Datentyp GeschenkRef *)

PROCEDURE ErzeugeGeschenk(beschreibung: TEXT): GeschenkRef=
(* Erzeugt ein Geschenk fuer den gegebenen Parameter  beschreibung
   auf der Halde und gibt den Zeiger darauf zurueck *)

VAR neuGeschenkRef: GeschenkRef;

BEGIN
  neuGeschenkRef := NEW(GeschenkRef);
  neuGeschenkRef^.beschreibung := beschreibung;
  neuGeschenkRef^.naechstes := NIL;

  RETURN neuGeschenkRef;
END ErzeugeGeschenk;

PROCEDURE SucheGeschenk(vorrat: GeschenkRef; beschreibung: TEXT): GeschenkRef=
(* Suche das Geschenk mit dem gegebenen Parameter  beschreibung *)

VAR aktGeschenkRef: GeschenkRef;

BEGIN
  (* Suche entlang der einfach verketteten Geschenkeliste *)
  aktGeschenkRef := vorrat;
  WHILE (aktGeschenkRef^.naechstes # NIL) AND
        (Text.Compare(aktGeschenkRef^.beschreibung, beschreibung) # 0)
  DO
    aktGeschenkRef := aktGeschenkRef^.naechstes;
  END;
```

```
  (* Falls Ende der Liste erreicht, ohne Geschenk zu finden *)
  IF (Text.Compare(aktGeschenkRef^.beschreibung, beschreibung) # 0) THEN
    aktGeschenkRef := NIL;
  END;

  RETURN aktGeschenkRef;
END SucheGeschenk;


PROCEDURE SucheVorgaengerGeschenk( vorrat: GeschenkRef;
                                   beschreibung: TEXT ): GeschenkRef=
(* Suche das vorhergehende Geschenk zum Geschenk mit dem gegebenen
   Parameter beschreibung *)

VAR aktGeschenkRef: GeschenkRef;
    vorGeschenkRef: GeschenkRef;

BEGIN
  aktGeschenkRef := vorrat;
  vorGeschenkRef := NIL;
  WHILE (aktGeschenkRef^.naechstes # NIL) AND
        (Text.Compare(aktGeschenkRef^.beschreibung, beschreibung) # 0)
  DO
    vorGeschenkRef := aktGeschenkRef;
    aktGeschenkRef := aktGeschenkRef^.naechstes;
  END;

  (* Falls Ende der Liste erreicht, ohne Geschenk zu finden *)
  IF (Text.Compare(aktGeschenkRef^.beschreibung, beschreibung) # 0) THEN
    vorGeschenkRef := NIL;
  END;

  RETURN vorGeschenkRef;
END SucheVorgaengerGeschenk;


PROCEDURE EinlagernGeschenk(VAR vorrat: GeschenkRef;
                            neuGeschenkRef: GeschenkRef)=
(* Lagere das Geschenk ein durch Anhaengen an die in  vorrat
   gegebene Geschenkliste *)

VAR aktGeschenkRef: GeschenkRef;

BEGIN
  (* Anhaengen des Geschenks an die Liste *)
  IF (vorrat = NIL) THEN
    vorrat := neuGeschenkRef;
  ELSE
    aktGeschenkRef := vorrat;
    WHILE (aktGeschenkRef^.naechstes # NIL) DO
      aktGeschenkRef := aktGeschenkRef^.naechstes;
    END;
    aktGeschenkRef^.naechstes := neuGeschenkRef;
  END;
END EinlagernGeschenk;
```

```
PROCEDURE EinlagernGeschenkDialog( lager: ArtRef)=
(* Fragt den Benutzer nach dem neu einzulagernden Geschenk und
   lagert dieses ein *)

VAR artName       : TEXT;
    beschreibung  : TEXT;
    neuGeschenkRef: GeschenkRef;
    aktArtRef     : ArtRef;

BEGIN
  (* Frage den Benutzer nach der Art des einzulagernden Geschenks *)
  SIO.PutLine("Einlagern eines Geschenks: ");
  artName := ErfrageArt();

  aktArtRef := SucheArt(lager, artName);
  IF (aktArtRef # NIL) THEN

    (* Frage Benutzer nach der Beschreibung fuer das Geschenk *)
    beschreibung := ErfrageBeschreibung();

    (* Erzeuge neues Geschenk mit diesen Daten *)
    neuGeschenkRef := ErzeugeGeschenk(beschreibung);
    EinlagernGeschenk(aktArtRef^.vorrat, neuGeschenkRef);

    (* Erhoehe Anzahl der vorhandenen Geschenke dieser Art um eins *)
    aktArtRef^.anzahl := aktArtRef^.anzahl + 1;

  ELSE
    SIO.PutLine("Eine solche Art gibt es nicht!");
  END;
END EinlagernGeschenkDialog;


PROCEDURE AuslagernGeschenk(aktArtRef: ArtRef; aktGeschenkRef: GeschenkRef)=
(* Nimm das in aktGeschenkRef referenzierte Geschenk aus der Liste *)
VAR vorGeschenkRef: GeschenkRef;

BEGIN
  (* Suche das Vorgaengergeschenk in der Liste *)
  vorGeschenkRef := SucheVorgaengerGeschenk(aktArtRef^.vorrat,
                                            aktGeschenkRef^.beschreibung);

  IF (vorGeschenkRef = NIL) THEN
    (* Loesche erstes Geschenk in der Liste *)
    aktArtRef^.vorrat := aktGeschenkRef^.naechstes;
  ELSE
    (* Loesche Geschenk in der Mitte *)
    vorGeschenkRef^.naechstes := aktGeschenkRef^.naechstes;
  END;

  (* Setze Anzahl Geschenke dieser Art um eins herab *)
  aktArtRef^.anzahl := aktArtRef^.anzahl - 1;
END AuslagernGeschenk;
```

```
PROCEDURE AuslagernGeschenkDialog(lager: ArtRef)=
(* Lagert das vom Benutzer eingegebene Geschenk aus *)

VAR artName, beschreibung: TEXT;
    aktArtRef            : ArtRef;
    aktGeschenkRef       : GeschenkRef;

BEGIN
  (* Frage Benutzer nach Art des auszulagernden Geschenks *)
  SIO.PutLine("Auslagern eines Geschenks: ");
  artName := ErfrageArt();

  aktArtRef := SucheArt(lager, artName);
  IF (aktArtRef # NIL) THEN

    (* Geschenkart existiert, pruefe, ob Geschenke dazu vorhanden *)
    IF (aktArtRef^.vorrat # NIL) THEN

      (* Gib Liste der vorhandenen Geschenke fuer die Geschenkart aus *)
      SIO.Nl();
      DruckeGeschenke(aktArtRef^.vorrat);
      SIO.Nl();

      (* Und frage Benutzer nach der Beschreibung des Geschenks *)
      beschreibung := ErfrageBeschreibung();

      (* Suche Geschenk und Vorgaenger mit der angegebenen Beschreibung *)
      aktGeschenkRef := SucheGeschenk(aktArtRef^.vorrat, beschreibung);

      IF (aktGeschenkRef # NIL) THEN

        (* Loesche Geschenk aus dem Lager *)
        AuslagernGeschenk(aktArtRef, aktGeschenkRef);
        SIO.PutLine("Das Geschenk wurde dem Lager entnommen!");

        IF NOT GeschenkAnzahlOK(aktArtRef) THEN
          SIO.PutLine("Achtung, Mindestzahl an Geschenken dieser Art " &
                      "unterschritten!");
        END;

      ELSE
        SIO.PutLine("Es gibt kein Geschenk mit dieser Beschreibung!");
      END;
    ELSE
      SIO.PutLine("Es gibt kein Geschenk dieser Art!");
    END;
  ELSE
    SIO.PutLine("Ein Geschenk einer solchen Art gibt es nicht!");
  END;
END AuslagernGeschenkDialog;


PROCEDURE DruckeGeschenke(vorrat: GeschenkRef)=
(* Druckt den gesamten Vorrat zu einer Geschenkart aus *)

VAR aktGeschenkRef: GeschenkRef;

BEGIN
  aktGeschenkRef := vorrat;
  WHILE (aktGeschenkRef # NIL) DO
    SIO.PutLine(aktGeschenkRef^.beschreibung);
    aktGeschenkRef := aktGeschenkRef^.naechstes;
  END;
END DruckeGeschenke;
```

```
(* Hauptprogramm *)

BEGIN
  (* Initialisiere das leere Geschenkelager *)
  InitLager(lager);

  REPEAT
    auswahl := ErfrageOperation();
    CASE auswahl OF
    | 'a' => AufnehmenArtDialog(lager);
    | 'n' => EinlagernGeschenkDialog(lager);
    | 'l' => AuslagernGeschenkDialog(lager);
    | 's' => DruckeLagerinhalt(lager);
    | 'e' => (* Mache nichts! *)
    ELSE
      SIO.PutLine("Keine gueltige Operation!");
    END;
  UNTIL (auswahl = 'e');

END Ruprecht101.
```

Ein möglicher Programmablauf (**mit Benutzereingaben**) sieht folgendermaßen aus:

```
Neue Geschenkart aufnehmen: a
Neues Geschenk einlagern  : n
Geschenk auslagern        : l
Geschenkstatistik erzeugen: s
Ende                      : e

Auswahl: a

Aufnehmen einer neuen Geschenkart:
Geschenkart: Lokomotive

Neue Geschenkart aufnehmen: a
...

Auswahl: n

Einlagern eines Geschenks:
Geschenkart: Lokomotive
Beschreibung des Geschenks: Dampflok

Neue Geschenkart aufnehmen: a
...

Auswahl: n

Einlagern eines Geschenks:
Geschenkart: Lokomotive
Beschreibung des Geschenks: Diesellok

Neue Geschenkart aufnehmen: a
...

Auswahl: n

Einlagern eines Geschenks:
Geschenkart: Lokomotive
Beschreibung des Geschenks:
  Elektrolok

Neue Geschenkart aufnehmen: a
...

Auswahl: a

Aufnehmen einer neuen Geschenkart:
Geschenkart: Schaukelpferd


Neue Geschenkart aufnehmen: a
...

Auswahl: n

Einlagern eines Geschenks:
Geschenkart: Schaukelpferd
Beschreibung des Geschenks: Rotes
  Schaukelpferd

Neue Geschenkart aufnehmen: a
...

Auswahl: s

Aktueller Lagerinhalt:

Lokomotive:
Dampflok
Diesellok
Elektrolok

Schaukelpferd:
Rotes Schaukelpferd
```

```
Neue Geschenkart aufnehmen: a
...

Auswahl: l

Auslagern eines Geschenks:
Geschenkart: Lokomotive

Dampflok
Diesellok
Elektrolok

Beschreibung des Geschenks: Diesellok
Das Geschenk wurde dem Lager
  entnommen!
Achtung, Mindestzahl an Geschenken
  dieser Art unterschritten!
```

```
Neue Geschenkart aufnehmen: a
...

Auswahl: s

Aktueller Lagerinhalt:

Lokomotive:
Dampflok
Elektrolok

Schaukelpferd:
Rotes Schaukelpferd

Neue Geschenkart aufnehmen: a
...

Auswahl: e
```

Lösung 10.2: Zählen von Wörtern

Das Wörter zählende Hauptprogramm:

```
MODULE Woerter102 EXPORTS Main;

(* Dieses Programm bestimmt alle Woerter und ihre jeweilige
   Anzahl in einer durch den Benutzer eingegebenen Textdatei
   und gibt das Ergebnis dann in einer ebenfalls vom Benutzer
   angegebenen Ausgabedatei aus.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 10.12.98  Letzte Aenderung: 04.01.98 *)

IMPORT SIO;         (* Importiere Operationen fuer Ein-/Ausgabe *)
IMPORT Text;        (* Importiere Operationen fuer Textmanipulation *)
IMPORT IO, Rd, Wr;  (* Importiere Operationen fuer Dateien *)
IMPORT Symbol;      (* Importiere die Wortscanner *)

TYPE WortRef = REF Wort;

     (* Doppelt verkettete Liste als Datenstruktur fuer Wortliste *)

     Wort = RECORD
              wort     : TEXT;
              anzahl   : CARDINAL;
              voriges  : WortRef;
              naechstes: WortRef;
            END;

VAR eingabeName, ausgabeName: TEXT;
    eingabe    : Rd.T;
    ausgabe    : Wr.T;
    woerter    : WortRef;  (* Anker fuer die Wortliste *)

(* Prozeduren zum Datentyp Wort aus dem die Wortliste aufgebaut  ist *)

PROCEDURE InitWoerter(VAR woerter: WortRef)=
(* Initialisiere Liste der Woerter mit leerer Liste *)
BEGIN
  woerter := NIL;
END InitWoerter;
```

```
PROCEDURE ErzeugeWort(wort: TEXT): WortRef=
(* Erzeugt fuer den gegebenen Parameter  wort auf der Halde
   einen Eintrag vom Typ Wort *)
VAR neuWortRef: WortRef;

BEGIN
  neuWortRef            := NEW(WortRef);
  neuWortRef^.wort      := wort;
  neuWortRef^.anzahl    := 1;
  neuWortRef^.voriges   := NIL;
  neuWortRef^.naechstes := NIL;

  RETURN neuWortRef;
END ErzeugeWort;

PROCEDURE ErhoeheAnzahlWort(aktWortRef: WortRef)=
(* Erhoehe die Anzahl des durch den Zeiger aktWortRef referenzierten
   Worts um eins *)
BEGIN
  aktWortRef^.anzahl := aktWortRef^.anzahl + 1;
END ErhoeheAnzahlWort;

PROCEDURE FuegeAlsVorgaengerEin(VAR woerter: WortRef;
                                aktWortRef, neuWortRef: WortRef )=
(* Fuege das durch den Zeiger neuWortRef referenzierte Wort vor dem
   durch den Zeiger aktWortRef referenzierten Eintrag in die durch
   den Parameter woerter als Anker repraesentierte Wortliste ein.
   Vorbedingung: Alle Parameter sind ungleich NIL!  *)
BEGIN
  (* Nachfolger des neuen Wortes ist der aktuelle Eintrag *)
  neuWortRef^.naechstes := aktWortRef;

  IF (aktWortRef^.voriges = NIL) THEN
    (* Anker aendern, da neues Wort am Anfang der Liste *)
    woerter := neuWortRef;
    neuWortRef^.voriges := NIL; (* = aktWortRef^.voriges *)
  ELSE
    (* Nachfolger des Vorgaengers des aktuellen Eintrags ist das neue Wort *)
    aktWortRef^.voriges^.naechstes := neuWortRef;
    neuWortRef^.voriges := aktWortRef^.voriges;
  END;

  (* Vorgaenger des aktuellen Eintrags ist das neue Wort *)
  aktWortRef^.voriges := neuWortRef;
END FuegeAlsVorgaengerEin;

PROCEDURE FuegeAlsNachfolgerEin(aktWortRef, neuWortRef: WortRef)=
(* Fuege das durch den Zeiger neuWortRef referenzierte Wort nach dem
   durch den Zeiger aktWortRef referenzierten Eintrag in die zugehoerige
   Wortliste ein. Vorbedingung: Alle Parameter sind ungleich NIL! *)
BEGIN
  (* Vorgaenger des neuen Worts ist der aktuelle Eintrag *)
  neuWortRef^.voriges := aktWortRef;
  (* Nachfolger des neuen Worts ist der Nachfolger des aktuellen Eintrags *)
  neuWortRef^.naechstes := aktWortRef^.naechstes;

  IF (aktWortRef^.naechstes # NIL) THEN
    (* Setze Vorgaenger des Nachfolgers des aktuellen Eintrags auf neues Wort *)
    aktWortRef^.naechstes^.voriges := neuWortRef;
  END;

  (* Nachfolger des aktuellen Eintrags ist das neue Wort *)
  aktWortRef^.naechstes := neuWortRef;
END FuegeAlsNachfolgerEin;
```

```
PROCEDURE FuegeWortEin(VAR woerter: WortRef; wort: TEXT)=
(* Fuegt das Wort an der entsprechenden Stelle (lexikographisch aufsteigend
   sortiert) in die Liste ein. *)

VAR aktWortRef: WortRef;
    neuWortRef: WortRef;
    vergleich : INTEGER;

BEGIN
  (* Initialisiere aktuellen Listeneintrag mit Anfang der Liste*)
  aktWortRef := woerter;
  (* Gedanklich sind alle Woerter vor Listenbeginn vor einzufuegendem Wort *)
  vergleich  := -1;

  IF (aktWortRef # NIL) THEN

    (* Suche das Wort bzw. seinen Nachfolger in der Liste *)
    vergleich  := Text.Compare(aktWortRef^.wort, wort);
    WHILE (aktWortRef^.naechstes # NIL) AND
          (vergleich < 0)
    DO
      aktWortRef := aktWortRef^.naechstes;
      vergleich  := Text.Compare(aktWortRef^.wort,  wort);
    END;

    IF (vergleich = 0) THEN
      (* Wort bereits in Liste enthalten, erhoehe nur Anzahl um eins *)
      ErhoeheAnzahlWort(aktWortRef);
    ELSE
      (* Wort noch nicht in Liste enthalten, erzeuge neuen Eintrag *)
      neuWortRef := ErzeugeWort(wort);

      IF (vergleich > 0) THEN
        (* Wort vor aktuellem Wort in Liste einfuegen *)
        FuegeAlsVorgaengerEin(woerter, aktWortRef, neuWortRef);
      ELSE (* vergleich < 0 *)
        (* Wort nach aktuellem Wort in Liste einfuegen *)
        FuegeAlsNachfolgerEin(aktWortRef, neuWortRef);
      END;
    END;
  ELSE (* aktWortRef = NIL *)
    (* Liste noch leer, setze Liste einfach auf neues Wort *)
    woerter := ErzeugeWort(wort);
  END;
END FuegeWortEin;
```

```
(* Prozeduren fuer die Bearbeitung der Dateien *)

PROCEDURE GibWoerterAus(wr: Wr.T; woerter: WortRef)=
(* Gib die Liste der Woerter in lexikographisch aufsteigender
   Reihenfolge in der durch wr referenzierten Datei aus *)

VAR aktWortRef: WortRef;
    anzahl    : CARDINAL;

BEGIN
  aktWortRef := woerter;

  anzahl := 0;
  WHILE (aktWortRef # NIL) DO
    SIO.PutText(aktWortRef^.wort & "   ", wr);
    SIO.PutInt(aktWortRef^.anzahl, 10, wr);
    SIO.Nl(wr);
    anzahl := anzahl + aktWortRef^.anzahl;
    aktWortRef := aktWortRef^.naechstes;
  END;
  SIO.PutInt(anzahl, 10, wr);
  SIO.Nl(wr);
END GibWoerterAus;


PROCEDURE ZaehleWoerter(rd: Rd.T; VAR woerter: WortRef)=
(* Ermittelt mit Hilfe des Moduls Symbol die Woerter in der
   durch rd referenzierten Eingabedatei und baut daraus eine
   Liste der Woerter mit ihrer jeweiligen Anzahl auf *)

VAR symbol: Symbol.T;

BEGIN
  (* Initialisiere den Wort-Scanner *)
  Symbol.Init(rd);

  (* Lese alle Symbole (Woerter) aus der Eingabedatei und
     fuege sie in die Liste der Woerter ein *)
  symbol := Symbol.LeseSymbol();
  WHILE (symbol.art # Symbol.SymbolArt.EOF) DO
    IF (symbol.art # Symbol.SymbolArt.Unbekannt) THEN
      FuegeWortEin(woerter, symbol.text);
    END;
    symbol := Symbol.LeseSymbol();
  END;
END ZaehleWoerter;

BEGIN
  InitWoerter(woerter);

  SIO.PutLine("Name der Eingabedatei: ");
  eingabeName := SIO.GetLine();
  SIO.PutLine("Name der Ausgabedatei: ");
  ausgabeName := SIO.GetLine();

  eingabe := IO.OpenRead(eingabeName);
  ausgabe := IO.OpenWrite(ausgabeName);

  ZaehleWoerter(eingabe, woerter);
  GibWoerterAus(ausgabe, woerter);

  Rd.Close(eingabe);
  Wr.Close(ausgabe);
END Woerter102.
```

Zusatzmodul Symbol als Wortscanner:

```
INTERFACE Symbol;

(* Diese Schnittstelle stellt Operationen und Datentypen fuer
   die Erkennung von Symbolen in einem Text bereit. Dazu wird
   eine reduzierte und veraenderte Version des Scanners aus
   Aufg. 9.1 verwendet.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 10.12.98  Letzte Aenderung: 04.01.99 *)

IMPORT Rd; (* Importiere Datentyp eines Eingabestroms aus  Rd *)

(* Datentypen fuer die uns interessierenden Symbole im Text *)

TYPE SymbolArt = {Wort, EOF,  Unbekannt};

     T         = RECORD
                   art    : SymbolArt;
                   text   : TEXT;      (* u.a. das Wort *)
                 END;

PROCEDURE Init(reader: Rd.T);
(* Initialisiere den Scanner, d.h. lese das erste Zeichen des Eingabestroms
   und setze den Eingabestrom auf den Parameter reader. *)

PROCEDURE LeseSymbol(): T;
(* Liest ein Symbol (Wort, Zahl, ...) aus dem gesetzten Eingabestrom. *)

END Symbol.


MODULE Symbol;

(* Dieses Modul stellt Operationen fuer die Erkennung von
   Symbolen (insbesondere Woerter) in einem Text bereit,
   implementiert also die gleichnamige Schnittstelle. Dazu
   wird eine reduzierte und veraenderte Version des Scanners
   aus Aufg. 9.1 verwendet.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 10.12.98  Letzte Aenderung: 04.01.99 *)


(* Private Deklarationen fuer die Implementierung *)

IMPORT SIO;  (* Importiere Ein-/Ausgabeoperationen *)
IMPORT Text; (* Importiere Operationen zur Textmanipulation *)
IMPORT IO;   (* Importiere Operationen fuer Dateien *)
IMPORT Rd;   (* Importiere Datentyp eines Eingabestroms *)

CONST Buchstaben       = SET OF CHAR {'a' .. 'z', 'A' .. 'Z'};
      Leerzeichen      = SET OF CHAR {' ', '\t', '\r', '\n'};

VAR zeichen: CHAR;   (* Globale Variable fuer zuletzt gelesenes Zeichen *)
    rd     : Rd.T;   (* Interne Referenz auf den Eingabestrom *)
```

```
PROCEDURE Init(reader: Rd.T)=
(* Initialisiere den Scanner, d.h. lese das erste Zeichen des Eingabestroms,
   falls moeglich, und setze den Eingabestrom auf den Parameter reader. * )
BEGIN
  rd      := reader;
  IF NOT IO.EOF(rd) THEN
    zeichen := SIO.GetChar(reader);
  END;
END Init;


PROCEDURE LeseSymbol(): T=
(* Liest ein Symbol (Wort, Satzzeichen, ...) aus dem gesetzten Eingabestrom. * )

VAR resultat: T;

BEGIN
  (* Ueberspringe Leerzeichen, solange Zeichen in der Datei vorhanden *)
  WHILE NOT IO.EOF(rd) AND  zeichen IN Leerzeichen DO
    zeichen := SIO.GetChar(rd);
  END;

  (* Anstehendes Zeichen bestimmt, welches Symbol vorliegt *)

  IF IO.EOF(rd) AND zeichen IN Leerzeichen THEN
    (* Ende der Datei erreicht und kein verwertbares Zeichen mehr,
       also gib Symbol EOF zurueck *)
    resultat.art  := SymbolArt.EOF;
    resultat.text := "EOF";
  ELSIF zeichen IN Buchstaben THEN
    (* Lese das Wort (nur Buchstaben) als Symbol ein *)
    resultat := LeseWort();
  ELSE
    (* Kein Leezeichen, aber auch kein Buchstabe, damit unbekanntes Symbol *)
    resultat.art  := SymbolArt.Unbekannt;
    resultat.text := "Unbekannt: " & Text.FromChar(zeichen);

    (* Versuche naechstes Zeichen aus Datei zu lesen *)
    IF NOT IO.EOF(rd) THEN
      zeichen := SIO.GetChar(rd);
    ELSE
      zeichen := ' ';  (* Setze naechstes Zeichen auf Leerzeichen *)
    END;
  END;

  RETURN resultat;
END LeseSymbol;


PROCEDURE LeseWort(): T=
(* Liest ein Wort (nur Buchstaben)  aus dem gesetzten Eingabestrom
   rd und erzeugt das entsprechende Symbol. * )

VAR resultat: T;

BEGIN
  resultat.art := SymbolArt.Wort;

  (* Lege das Wort in der Komponente text ab *)
  resultat.text := "";
  WHILE NOT IO.EOF(rd) AND (zeichen IN Buchstaben) DO
    resultat.text := resultat.text & Text.FromChar(zeichen);
    zeichen := SIO.GetChar(rd);
  END;
```

```
  (* Beruecksichtige letzten Buchstaben vor dem Dateiende *)
  IF IO.EOF(rd) AND (zeichen IN Buchstaben) THEN
    resultat.text := resultat.text & Text.FromChar(zeichen);
    zeichen := ' '; (* Setze Leerzeichen als naechstes Zeichen *)
  END;

  RETURN resultat;
END LeseWort;


BEGIN
  (* Initialisiere interne Variablen *)
  zeichen := ' ';
  rd := NIL;
END Symbol.
```

Die Ergebnisdatei für den Beispieltext "Ribbeck.txt" sieht aus, wie folgt:

```
Aber   1
Ach   1
Alle   1
Als   1
Bauern   1
Baume   1
Beer   3
Birn   3
Birnbaum   3
Birnbaumsproessling
1
Birne   1
Birnen   2
Buedner   1
Da   2
Das   1
Der   4
Des   1
Dirn   2
Doppeldachhaus   1
Ein   2
Ende   1
Er   1
Feiergesicht   1
Fontane   1
Garten   1
Grab   3
Haelt   1
Hand   1
Haus   1
Havelland   3
He   1
Herbsteszeit   3
Herr   2
Herze   1
Ich   1
Jahr   1
Jahre   2
Jesus   1
Jung   1
Junge   2
Kinder   2
Kirchhof   1
Kumm   2
Laengst   1
Legt   1
Leuchtet   1
Luett   2
Maedel   2
Misstraun   1
Mittag   1
Pantinen   1
Park   1
Ribbeck   12
Sangen   1
Segen   1
So   5
Sohn   1
Tage   1
Taschen   1
Theodor   1
Trugen   1
Turme   1
Und   12
Wer   1
Wieder   1
Wiste   1
Zuversicht   1
ab   2
alte   1
alten   1
auf   5
aus   2
bat   1
beide   1
bis   1
breit   3
daher   1
damals   1
das   1
dem   3
den   2
der   3
di   1
die   7
dod   1
drauf   1
drei   1
dritten   1
eigenen   1
ein   5
eine   2
er   4
es   1
fluestert   2
freilich   1
fuehlte   1
gegen   1
genau   1
gew   1
ging   1
gingen   1
giwt   1
goldene   1
goldenen   1
hebb   1
her   1
heraus   1
hinaus   1
ick   2
im   5
immer   1
in   3
ins   2
is   1
kam   4
kannten   1
klagten   2
knausert   1
kommt   2
lachten   1
leuchteten   1
lobesam   1
man   2
meine   1
mir   1
mit   2
ne   5
neue   1
nicht   1
noch   1
nu   2
nun   1
recht   1
rief   2
roewer   2
s   5
sagte   1
scheide   1
schlecht   1
scholl   1
schon   1
schwer   1
sein   1
seinem   1
sich   2
sie   2
so   2
spart   1
spendet   1
sprosst   1
stand   1
sterben   1
stillen   1
stopfte   1
strenge   1
tat   1
ueber   1
uebern   1
um   1
und   7
uns   1
verwahrt   1
viel   1
voll   2
vom   1
von   7
vorahnend   1
war   2
was   1
weit   3
wenn   1
wieder   1
wiste   1
woelbt   1
wohl   1
wusste   1
zu   1

Gesamtzahl: 302
```

Lösungen zu Übung 11

Lösung 11.1: Matrix mit Prozedurtypen

```
MODULE Matrix111 EXPORTS Main;

(* Dieses Program erzeugt verschiedene Drehungen und Spiegelungen einer
   quadratischen Matrix. Alternative Loesung zu Aufg. 8.1 mit Prozedurtypen.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 06.01.99 *)


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


CONST MaxElemente = 10; (* Maximalzahl Elemente pro Reihe/Spalte der Matrix *)


TYPE Index = [1..MaxElemente];

     Matrix = ARRAY Index, Index OF CHAR;

     Position = RECORD
                  i: Index;
                  j: Index;
                END;

      (* Prozedurtyp fuer die Berechnung der Elementpositionen in der Matrix *)

      PositionProz = PROCEDURE (i, j: Index): Position;


CONST
  (* Initialisiere konstante Zeilen der Matrix mit dem vorgegebenen Muster *)
  Zeile1  = ARRAY Index OF CHAR {'*','*','*','*','*','*','*','*','*','*' };
  Zeile2  = ARRAY Index OF CHAR {' ',' ',' ',' ',' ',' ',' ',' ','*',' ' };
  Zeile3  = ARRAY Index OF CHAR {' ',' ',' ','*',' ',' ',' ','*',' ',' ' };
  Zeile4  = ARRAY Index OF CHAR {' ',' ',' ',' ',' ',' ','*',' ',' ',' ' };
  Zeile5  = ARRAY Index OF CHAR {' ',' ',' ',' ',' ','*',' ',' ',' ',' ' };
  Zeile6  = ARRAY Index OF CHAR {' ',' ',' ',' ','*',' ',' ',' ',' ',' ' };
  Zeile7  = ARRAY Index OF CHAR {' ',' ',' ','*',' ',' ',' ',' ',' ',' ' };
  Zeile8  = ARRAY Index OF CHAR {' ',' ','*',' ',' ',' ',' ',' ',' ',' ' };
  Zeile9  = ARRAY Index OF CHAR {' ','*',' ',' ',' ',' ',' ',' ',' ',' ' };
  Zeile10 = ARRAY Index OF CHAR {'*','*','*','*','*','*','*','*','*','*' };

VAR matrix   := Matrix { Zeile1, Zeile2, Zeile3, Zeile4, Zeile5,
                         Zeile6, Zeile7, Zeile8, Zeile9, Zeile10};
    ergMatrix: Matrix;
    operation: CHAR;
    dummy    : TEXT;
```

```
(* Implementierungen zum Prozedurtyp PositionProz *)

PROCEDURE SpiegleVertSymAchse(i, j: Index): Position=
(* Spiegelt Element an der vertikalen Symmetrieachse *)
VAR resultat: Position;

BEGIN
  resultat.i := i;
  resultat.j := LAST(Index) - j + 1;

  RETURN resultat;
END SpiegleVertSymAchse;
PROCEDURE SpiegleHoriSymAchse(i, j: Index): Position=
(* Spiegelt Element an der horizontalen Symmetrieachse *)
VAR resultat: Position;

BEGIN
  resultat.i := LAST(Index) - i + 1;
  resultat.j := j;

  RETURN resultat;
END SpiegleHoriSymAchse;


PROCEDURE SpiegleHauptdiagonale(i, j: Index): Position=
(* Spiegelt Element an der Hauptdiagonalen *)
VAR resultat: Position;

BEGIN
  resultat.i := j;
  resultat.j := i;

  RETURN resultat;
END SpiegleHauptdiagonale;


PROCEDURE SpiegleNebendiagonale(i, j: Index): Position=
(* Spiegelt Element an der Nebendiagonalen *)
VAR resultat: Position;

BEGIN
  resultat.i := LAST(Index) - j + 1;
  resultat.j := LAST(Index) - i + 1;

  RETURN resultat;
END SpiegleNebendiagonale;


PROCEDURE DreheImUhrzeigersinn(i, j: Index): Position=
(* Dreht Element im Uhrzeigersinn *)
VAR resultat: Position;

BEGIN
  resultat.i := LAST(Index) - j + 1;
  resultat.j := i;

  RETURN resultat;
END DreheImUhrzeigersinn;
```

```
PROCEDURE DreheGegenUhrzeigersinn(i, j: Index): Position=
(* Dreht Element gegen den Uhrzeigersinn *)
VAR resultat: Position;

BEGIN
  resultat.i := j;
  resultat.j := LAST(Index) - i + 1;

  RETURN resultat;
END DreheGegenUhrzeigersinn;


PROCEDURE Identitaet(i, j: Index): Position=
(* Bildet ein Element auf sich selbst ab *)
VAR resultat: Position;

BEGIN
  resultat.i := i;
  resultat.j := j;

  RETURN resultat;
END Identitaet;


PROCEDURE BestimmeMatrix(READONLY matrix: Matrix;
                         VAR ergMatrix: Matrix; operation: PositionProz)=
(* Bestimmt die aufgrund des gegebenen Parameters operation umgespeicherte
   Ergebnismatrix ergMatrix aus der gegebenen Ursprungsmatrix matrix *)

VAR urPosition: Position;

BEGIN
  FOR i := 1 TO LAST(Index) DO
    FOR j := 1 TO LAST(Index) DO
      urPosition := operation(i, j);  (* Aufruf der Parameter-Prozedur *)
      ergMatrix[i, j] := matrix[urPosition.i, urPosition.j];
    END;
  END;
END BestimmeMatrix;


PROCEDURE DruckeMatrix(READONLY matrix: Matrix)=
(* Gibt die gegebene Matrix reihenweise aus *)
BEGIN
  FOR i := 1 TO LAST(Index) DO
    FOR j := 1 TO LAST(Index) DO
      SIO.PutChar(matrix[i, j]);
    END;
    SIO.Nl();
  END;
END DruckeMatrix;


BEGIN

  REPEAT
    SIO.PutLine("Operationen zur Manipulation der Matrix: " &
                "v, h, d, n, i, g. e = Ende!");
    SIO.PutText("Auswahl: ");

    (* Lies Eingabe und RETURN *)
    operation := SIO.GetChar();
    dummy := SIO.GetLine();     (* Entfernt RETURN aus der Eingabe *)
```

```
    IF (operation # 'e') THEN

      CASE operation OF
      | 'v' => BestimmeMatrix(matrix, ergMatrix, SpiegleVertSymAchse);
      | 'h' => BestimmeMatrix(matrix, ergMatrix, SpiegleHoriSymAchse);
      | 'd' => BestimmeMatrix(matrix, ergMatrix, SpiegleHauptdiagonale);
      | 'n' => BestimmeMatrix(matrix, ergMatrix, SpiegleNebendiagonale);
      | 'i' => BestimmeMatrix(matrix, ergMatrix, DreheImUhrzeigersinn);
      | 'g' => BestimmeMatrix(matrix, ergMatrix, DreheGegenUhrzeigersinn);
      ELSE
        BestimmeMatrix(matrix, ergMatrix, Identitaet);
      END;

      SIO.Nl();
      DruckeMatrix(ergMatrix);
      SIO.Nl();
    END;
  UNTIL operation = 'e';

END Matrix111.
```

Lösung 11.2: Tennisrangliste als Objektmodul

Das Hauptprogramm für die Interaktion mit dem Benutzer:

```
MODULE Tennis112 EXPORTS Main;

(* Dieses Programm erlaubt die Verwaltung einer Tennisrangliste, wie in Aufg.
   9.2. Hier werden jedoch die Implementierungen aus dem Objektmodul Tennis und
   des ADTs Spieler verwendet.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 07.01.99 *)


IMPORT SIO;      (* Importiere Operationen fuer die Ein-/Ausgabe *)
IMPORT Spieler;  (* Importiere den ADT Spieler *)
IMPORT Tennis;   (* Importiere das Objektmodul Tennis *)

VAR spieler, gewinner, verlierer: Spieler.T; (* Variablen des ADTs Spieler *)
    auswahl                     : CHAR;
    dummy                       : TEXT;


PROCEDURE DruckeMenue()=
(* Hilfsprozedur, die das Befehlsmenue ausgibt *)
BEGIN
  SIO.Nl();
  SIO.PutLine("F : Fuege einen Spieler ein");
  SIO.PutLine("L : Loesche einen Spieler");
  SIO.PutLine("G : Spiel gemacht");
  SIO.PutLine("Z : Zeige Rangliste an");
  SIO.PutLine("E : Exit");
  SIO.Nl();
END DruckeMenue;
```

```
BEGIN
  REPEAT
    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' => spieler := Spieler.LeseEin("Spieler Einfuegen");
                   SIO.Nl();
                   IF NOT Tennis.SpielerExistiertRL(spieler) THEN
                     IF NOT Tennis.VollRL() THEN
                       Tennis.EinfuegenSpielerRL(spieler);
                     ELSE
                       SIO.PutLine("Kein Platz mehr in der Rangliste!");
                     END;
                   ELSE
                     SIO.PutLine("Der Spieler existiert bereits in der " &
                                 "Rangliste!");
                   END;

    |  'L', 'l' => spieler := Spieler.LeseEin("Spieler Loeschen");
                   SIO.Nl();
                   IF Tennis.SpielerExistiertRL(spieler) THEN
                     Tennis.LoescheSpielerRL(spieler);
                   ELSE
                     SIO.PutLine("Der Spieler ist nicht in der Rangliste!");
                   END;




    |  'G', 'g' => gewinner := Spieler.LeseEin("Sieger des Spiels");
                   SIO.Nl();
                   verlierer := Spieler.LeseEin("Verlierer des Spiels");
                   SIO.Nl();

                   IF Tennis.SpielerExistiertRL(gewinner) AND
                      Tennis.SpielerExistiertRL(verlierer)
                   THEN
                     Tennis.AktualisiereRL(gewinner, verlierer);
                   ELSE
                     SIO.PutLine("Wenigstens einer der Spieler ist nicht in " &
                                 "der Rangliste!");
                   END;

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

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

  UNTIL (auswahl = 'e') OR (auswahl = 'E');
END Tennis112.
```

Schnittstelle des Objektmoduls, das die Tennisrangliste implementiert:

```
INTERFACE Tennis;

(* Schnittstelle des Objektmoduls zur Verwaltung einer Tennisrangliste.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 07.01.99 *)

IMPORT Spieler;  (* Importiere den ADT Spieler *)

PROCEDURE VollRL(): BOOLEAN;
(* Prueft, ob noch Platz fuer wenigstens einen Spieler in der Rangliste ist. *)


PROCEDURE SpielerExistiertRL(spieler: Spieler.T): BOOLEAN;
(* Prueft, ob der angegebene Spieler in der Rangliste enthalten ist. *)


PROCEDURE EinfuegenSpielerRL(spieler: Spieler.T);
(* Fuegt den gegebenen Spieler am Ende der Rangliste ein. Vorbedingung:
   Der Spieler ist noch nicht in der Rangliste enthalten und die Rangliste
   ist noch nicht voll. *)


PROCEDURE LoescheSpielerRL(spieler: Spieler.T);
(* Loescht den gegebenen Spieler aus der Rangliste. Vorbedingung:
   Der Spieler ist in der Rangliste enthalten. *)


PROCEDURE AktualisiereRL(gewinner, verlierer: Spieler.T);
(* Aendert die Rangfolge in der Rangliste entsprechend dem gegebenen
   Spielergebnis (Gewinner, Verlierer). Vorbedingung: Gewinner und
   Verlierer sind in der Rangliste enthalten  *)


PROCEDURE DruckeRL();
(* Gibt die Rangliste in absteigender Rangfolge aus *)

END Tennis.
```

Implementierung des Objektmoduls "Tennis" mit einer einfach verketteten Liste:

```
MODULE TennisListe EXPORTS Tennis;

(* Dieses Modul implementiert die Verwaltung einer Tennisrangliste,
   wie in der Schnittstelle Tennis definiert, mit Hilfe einer
   einfach verketteten Liste.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 06.01.99 *)


IMPORT SIO;      (* Importiere Operationen fuer die Ein-/Ausgabe *)
IMPORT Spieler;  (* Importiere den ADT Spieler *)


TYPE RanglisteRef = REF RanglistenElement;

     RanglistenElement = RECORD
                           spieler  : Spieler.T;
                           naechster: RanglisteRef;
                         END;

VAR rangliste    : RanglisteRef; (* Anker der Club-Rangliste *)


PROCEDURE SucheSpielerRL(spieler: Spieler.T): RanglisteRef=
(* Sucht den gegebenen Spieler in der Rangliste und gibt einen Zeiger auf ihn
   zurueck. Der Zeiger hat den Wert NIL, wenn der Spieler nicht in der Liste
   vorhanden ist. *)
VAR resRLRef: RanglisteRef;
    gefunden: BOOLEAN;

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

     (* Pruefe, ob der referenzierte Spieler der gesuchte ist *)
     IF Spieler.Gleich(resRLRef^.spieler, spieler) THEN
       gefunden := TRUE;
     ELSE
       resRLRef := resRLRef^.naechster;
     END;

   END; (* WHILE *)

   RETURN resRLRef;
END SucheSpielerRL;


PROCEDURE SucheVorgaengerRL(spieler: Spieler.T): RanglisteRef=
(* Sucht den Vorgaenger des gegebenen Spielers in der Rangliste und gibt
   einen Zeiger auf ihn zurueck. Der Zeiger hat den Wert NIL, wenn der
   Spieler 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 := rangliste;
```

```
   WHILE (aktRLRef # NIL) AND NOT gefunden DO
     (* Pruefe, ob der referenzierte Spieler der gesuchte ist *)
     IF Spieler.Gleich(aktRLRef^.spieler, spieler) THEN
       gefunden := TRUE;
     ELSE
       vorRLRef := aktRLRef;
       aktRLRef := aktRLRef^.naechster;
     END;
   END; (* WHILE *)
   (* Zeiger auf den Vorgaenger ist NIL, wenn Spieler nicht in der Liste ist *)
   IF NOT gefunden THEN vorRLRef := NIL; END;

   RETURN vorRLRef;
END SucheVorgaengerRL;


PROCEDURE VollRL(): BOOLEAN=
(* Prueft, ob noch Platz fuer wenigstens einen Spieler in der Rangliste ist. *)
BEGIN
  (* In der optimistischen Annahme wir haetten unbegrenzten Speicherplatz, ist
     die Rangliste nie voll *)
  RETURN FALSE;
END VollRL;


PROCEDURE SpielerExistiertRL(spieler: Spieler.T): BOOLEAN=
(* Prueft, ob der angegebene Spieler in der Rangliste enthalten ist. *)
BEGIN
  RETURN (SucheSpielerRL(spieler) # NIL);
END SpielerExistiertRL;


PROCEDURE EinfuegenSpielerRL(spieler: Spieler.T)=
(* Fuegt den gegebenen Spieler am Ende der Rangliste ein. Vorbedingung:
   Der Spieler ist noch nicht in der Rangliste enthalten. *)
VAR ranglisteRef, aktRLRef: RanglisteRef;

BEGIN
  (* Erzeuge neues Ranglisten-Element fuer den Spieler *)
  ranglisteRef := NEW(RanglisteRef);
  ranglisteRef^.spieler := spieler;
  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 Spieler ans Ende an *)
    aktRLRef^.naechster := ranglisteRef;
  END;
END EinfuegenSpielerRL;
```

```
PROCEDURE LoescheSpielerRL(spieler: Spieler.T)=
(* Loescht den gegebenen Spieler aus der Rangliste. Vorbedingung:
   Der Spieler ist in der Rangliste enthalten. *)
VAR ranglisteRef,
    vorgaengerRef: RanglisteRef;

BEGIN
  (* Suche den Spieler in der Liste *)
  ranglisteRef := SucheSpielerRL(spieler);

  IF (ranglisteRef # NIL) THEN

    (* Vorhandenen Spieler aus der Rangliste loeschen *)
    vorgaengerRef := SucheVorgaengerRL(spieler);
    IF vorgaengerRef = NIL THEN
      (* Erstes Element in Rangliste, daher kein Vorgaenger *)
      rangliste := ranglisteRef^.naechster;
    ELSE
      vorgaengerRef^.naechster := ranglisteRef^.naechster;
    END;
  END;
END LoescheSpielerRL;


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(gewinner, verlierer: Spieler.T)=
(* Aendert die Rangfolge in der Rangliste entsprechend dem gegebenen
   Spielergebnis (Gewinner, Verlierer). Vorbedingung: Gewinner und
   Verlierer sind in der Rangliste enthalten  *)

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

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

  IF NOT (gewinnerRef = NIL) AND NOT (verliererRef = NIL) THEN

    (* Pruefe, ob nicht Gewinner ohnehin vor Verlierer in der Rangliste! *)
    IF AKommtHinterBInRL(gewinnerRef, verliererRef) THEN
      (* Fuege Gewinner vor Verlierer in Rangliste ein *)

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

```
      (* Haenge Gewinner ein bei Vorgaenger des Verlierers *)
      IF ( vorVerliererRef = NIL) THEN
        (* Kein Vorgaenger, da ganz am Anfang der Liste *)
        rangliste := gewinnerRef;
      ELSE
        vorVerliererRef^.naechster := gewinnerRef;
      END;

      (* Haenge Nachfolger des Gewinners bei seinem Vorgaenger ein.
         BEACHTE: Wenigstens der Verlierer kommt hier vor dem Gewinner in der
         Liste, so dass die Referenz vorGewinnerRef nicht NIL sein kann. *)
      vorGewinnerRef^.naechster := gewinnerRef^.naechster;

      (* Haenge Verlierer als Nachfolger des Gewinners ein *)
      gewinnerRef^.naechster := verliererRef;
    END;

  END;
END AktualisiereRL;


PROCEDURE DruckeRL()=
(* Gibt die Rangliste in absteigender Rangfolge aus *)

VAR ranglisteRef: RanglisteRef;
    rang        : CARDINAL;

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


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

    (* Gib Rang und Daten des Spielers aus *)
    SIO.PutInt(rang); SIO.PutText(". ");
    Spieler.Drucke(ranglisteRef^.spieler);

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

  END; (* WHILE *)
END DruckeRL;


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

Implementierung des Objektmoduls "Tennis" mit einem Feld fester Größe als Datenstruktur:

```
MODULE TennisFeld EXPORTS Tennis;

(* Dieses Modul implementiert die Verwaltung einer Tennisrangliste,
   wie in der Schnittstelle Tennis definiert, mit Hilfe eines
   Feldes mit begrenzter Kapazitaet.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 06.01.99 *)


IMPORT SIO;      (* Importiere Operationen fuer Ein-/Ausgabe *)
IMPORT Spieler;  (* Importiere den ADT Spieler *)


CONST MaxAnzahlSpieler = 5;  (* die in der Rangliste Platz haben *)


TYPE RanglisteIndex = [0 .. MaxAnzahlSpieler];


VAR ranglisteIndex: RanglisteIndex;
    rangliste     : ARRAY [1..MaxAnzahlSpieler] OF Spieler.T;


PROCEDURE SucheSpielerRL(spieler: Spieler.T): RanglisteIndex=
(* Sucht den gegebenen Spieler in der Rangliste und gibt seinen Index
   zurueck. Der Index hat den Wert 0, wenn der Spieler nicht im Feld
   enthalten ist. *)

VAR resultat: RanglisteIndex;
    i       : INTEGER;

BEGIN
  i := 1;
  resultat := 0;
  WHILE (i <= ranglisteIndex) AND (resultat = 0) DO
    (* Pruefe, ob der indizierte Spieler der gesuchte ist. *)
    IF Spieler.Gleich(rangliste[i], spieler) THEN
      resultat := i;
    ELSE
      i := i + 1;
    END;
  END; (* WHILE *)

  RETURN resultat;
END SucheSpielerRL;


PROCEDURE VollRL(): BOOLEAN=
(* Prueft, ob noch Platz fuer wenigstens einen Spieler in der Rangliste ist. *)
BEGIN
  RETURN (ranglisteIndex = MaxAnzahlSpieler);
END VollRL;


PROCEDURE SpielerExistiertRL(spieler: Spieler.T): BOOLEAN=
(* Prueft, ob der angegebene Spieler in der Rangliste enthalten ist. *)
BEGIN
  RETURN (SucheSpielerRL(spieler) # 0);
END SpielerExistiertRL;
```

```
PROCEDURE EinfuegenSpielerRL(spieler: Spieler.T)=
(* Fuegt den gegebenen Spieler am Ende der Rangliste ein. Vorbedingung:
   Der Spieler ist noch nicht in der Rangliste enthalten und die Rangliste
   ist noch nicht voll. *)

BEGIN
  IF (ranglisteIndex < MaxAnzahlSpieler) THEN
    (* Noch Platz in der Rangliste *)
    ranglisteIndex := ranglisteIndex + 1;
    rangliste[ranglisteIndex] := spieler;
  END;
END EinfuegenSpielerRL;


PROCEDURE LoescheSpielerRL(spieler: Spieler.T)=
(* Loescht den gegebenen Spieler aus der Rangliste. Vorbedingung:
   Der Spieler ist in der Rangliste enthalten. *)

VAR spielerIndex: RanglisteIndex;

BEGIN
  (* Suche den Spieler in der Liste *)
  spielerIndex := SucheSpielerRL(spieler);

  IF (spielerIndex # 0) THEN

    (* Speichere nachfolgende Spieler je einen Platz vorher *)
    FOR i := spielerIndex TO ranglisteIndex DO
      IF (i < MaxAnzahlSpieler) THEN
        rangliste[i] := rangliste[i + 1];
      END;
    END;

    (* Verringere Anzahl der Ranglistenelemente um eins *)
    ranglisteIndex := ranglisteIndex - 1;
  END;
END LoescheSpielerRL;


PROCEDURE AktualisiereRL(gewinner, verlierer: Spieler.T)=
(* Aendert die Rangfolge in der Rangliste entsprechend dem gegebenen
   Spielergebnis (Gewinner, Verlierer). Vorbedingung: Gewinner und
   Verlierer sind in der Rangliste enthalten  *)

VAR gewinnerIndex, verliererIndex: RanglisteIndex;
    hilfsSpieler                 : Spieler.T;

BEGIN
  (* Suche zunaechst Gewinner und Verlierer in der Rangliste *)
  gewinnerIndex := SucheSpielerRL(gewinner);
  verliererIndex := SucheSpielerRL(verlierer);

  IF NOT (gewinnerIndex = 0) AND NOT (verliererIndex = 0) THEN

    (* Pruefe, ob nicht Gewinner ohnehin vor Verlierer in der Rangliste! *)
    IF (verliererIndex < gewinnerIndex) THEN


      (* Fuege Gewinner an Stelle des Verlierers in Rangliste ein *)
      hilfsSpieler := rangliste[verliererIndex];
      rangliste[verliererIndex] := rangliste[gewinnerIndex];
```

```
      (* Bewege alle Spieler zwischen Verlierer und Gewinner eine Position
         nach hinten *)
      FOR i := gewinnerIndex - 1 TO verliererIndex + 1 BY -1 DO
        rangliste[i + 1] := rangliste[i];
      END;

      (* Setze Verlierer eine Position hinter den Gewinner *)
      rangliste[verliererIndex + 1] := hilfsSpieler;
    END;
  END;

END AktualisiereRL;


PROCEDURE DruckeRL()=
(* Gibt die Rangliste in absteigender Rangfolge aus *)

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

  FOR rang := 1 TO ranglisteIndex DO
    (* Gib Rang und Daten des Spielers aus *)
    SIO.PutInt(rang); SIO.PutText(". ");
    Spieler.Drucke(rangliste[rang]);
  END;

END DruckeRL;


BEGIN
  (* Initialisiere Feld mit leerer Rangliste *)
  ranglisteIndex := 0;
END TennisFeld.
```

Schnittstelle und Implementierung des ADTs Spieler:

```
INTERFACE Spieler;

(* Schnittstelle eines ADTs fuer die Daten eines Vereinstennisspielers.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 07.01.99 *)


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

PROCEDURE LeseEin(nachricht: TEXT): T;
(* Fragt den Benutzer nach den Daten eines Spielers und gibt
   anschliessend einen entsprechenden Spieler zurueck. *)


PROCEDURE Drucke(spieler: T);
(* Gibt die Daten des Spielers aus. *)


PROCEDURE Gleich(spielerA, spielerB: T): BOOLEAN;
(* Prueft, ob die beiden Spieler gleich sind. *)

END Spieler.
```

```
MODULE Spieler;

(* Implementiert den ADT fuer die Daten eines Vereinstennis-
   spielers.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 06.01.99  Letzte Aenderung: 07.01.99 *)

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 Spieler *)

     REVEAL T = BRANDED REF RECORD
                  name       : Name;
                  mitgliedsNr: CARDINAL;
                END;

PROCEDURE LeseEin(nachricht: TEXT): T=
(* Fragt den Benutzer nach den Daten eines Spielers und gibt
   anschliessend einen entsprechenden Spieler zurueck. *)
VAR resultat: T;
    dummy   : TEXT;

BEGIN
  (* Erzeuge einen neuen Spieler 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("Mitgliedsnr: ");
  resultat^.mitgliedsNr := SIO.GetInt();
  dummy := SIO.GetLine();  (* Lies RETURN nach Mitgliedsnr. aus der Eingabe! *)
  RETURN resultat;
END LeseEin;

PROCEDURE Drucke(spieler: T)=
(* Gibt die Daten des Spielers aus. *)
BEGIN
  SIO.PutText(spieler^.name.nachname & "  ");
  SIO.PutText(spieler^.name.vorname & "  ");
  SIO.PutText("Mitglieds-Nr. ");
  SIO.PutInt(spieler^.mitgliedsNr); SIO.Nl();
END Drucke;

PROCEDURE Gleich(spielerA, spielerB: T): BOOLEAN=
(* Prueft, ob die beiden Spieler gleich sind. *)
BEGIN
  RETURN ((Text.Equal(spielerA^.name.vorname, spielerB^.name.vorname)) AND
          (Text.Equal(spielerA^.name.nachname, spielerB^.name.nachname)) AND
          (spielerA^.mitgliedsNr = spielerB^.mitgliedsNr));
END Gleich;

BEGIN
END Spieler.
```

Lösungen zu Übung 12

Lösung 12.1: Datenbank für Personendaten

```
MODULE Datenbank121 EXPORTS Main;

(* Dieses Programm verwaltet eine Liste mit Personendaten und
   bestimmt mit Hilfe einer mit einem Prozedtyp-Parameter
   versehenen Prozedur verschiedene Teillisten dieser Liste.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 07.01.98  Letzte Aenderung: 07.01.98 *)

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

TYPE Person  = RECORD
                 name      : TEXT;
                 vorname   : TEXT;
                 alter     : [0 .. 150];
               END;

     (* Einfach verkettete Liste als Datenstruktur fuer Personenliste *)

     ListeRef = REF ListenElement;

     ListenElement = RECORD
                       person   : Person;
                       naechstes: ListeRef;
                     END;

     (* Prozedurtyp fuer die Auswahl von Personen *)

     SelektProz = PROCEDURE (person: Person): BOOLEAN;


VAR liste      : ListeRef;  (* Anker fuer die Personenliste *)
    ergListe   : ListeRef;
    neuListeRef: ListeRef;
    neuPerson  : Person;


(* Prozeduren zum Datentyp Liste aus dem die Personenliste aufgebaut  ist *)

PROCEDURE InitListe(VAR liste: ListeRef)=
(* Initialisiert Personenliste mit leerer Liste. *)
BEGIN
  liste := NIL;
END InitListe;
```

```
PROCEDURE ErzeugeElementListe(person: Person): ListeRef=
(* Erzeugt fuer den gegebenen Parameter person auf der Halde
   einen Eintrag vom Typ ListeRef. *)

VAR neuListeRef: ListeRef;

BEGIN
  neuListeRef            := NEW(ListeRef);
  neuListeRef^.person    := person;
  neuListeRef^.naechstes := NIL;

  RETURN neuListeRef;
END ErzeugeElementListe;

PROCEDURE HaengeAnListe(VAR liste: ListeRef; element: ListeRef)=
(* Haengt das gegebene Listenelement ans Ende der Personenliste liste. * )

VAR aktListeRef: ListeRef;

BEGIN
  IF (liste = NIL) THEN
    (* Liste noch leer *)
    liste := element;
  ELSE
    (* Suche Ende der Liste *)
    aktListeRef := liste;
    WHILE (aktListeRef^.naechstes # NIL) DO
      aktListeRef := aktListeRef^.naechstes;
    END;

    (* Haenge Element an *)
    aktListeRef^.naechstes := element;
  END;
END HaengeAnListe;


PROCEDURE IteriereListe(liste: ListeRef; selekt: SelektProz): ListeRef=
(* Iteriert elementweise ueber die Personenliste liste und prueft mit
   Hilfe der Parameter-Prozedur selekt, welche davon in die Ergebnisliste
   uebernommen werden. Am Ende wird die Ergebnisliste zurueckgegeben. *)

VAR ergListe    : ListeRef;
    aktListeRef : ListeRef;
    neuListeRef : ListeRef;

BEGIN
  aktListeRef := liste;
  WHILE (aktListeRef # NIL) DO
    (* Bestimme Element durch Aufruf des Prozedur-Parameters. *)
    IF selekt(aktListeRef^.person) THEN
      neuListeRef := ErzeugeElementListe(aktListeRef^.person);
      HaengeAnListe(ergListe, neuListeRef);
    END;
    aktListeRef := aktListeRef^.naechstes;
  END;

  RETURN ergListe;
END IteriereListe;
```

```
PROCEDURE DruckeListe(liste: ListeRef)=
(* Druckt alle Elemente der gegebenen Liste aus *)

VAR aktListeRef: ListeRef;

BEGIN
  aktListeRef := liste;
  WHILE (aktListeRef # NIL) DO
    DruckePerson(aktListeRef^.person);
    aktListeRef := aktListeRef^.naechstes;
  END;
END DruckeListe;


(* Operationen zum Datentyp Person *)

PROCEDURE LesePerson(): Person=
(* Fragt den Benutzer nach den Daten zu einer Person und
   gibt zum Schluss die entsprechende Person zurueck. * )

VAR resultat: Person;
    dummy   : TEXT;

BEGIN
  SIO.PutLine("Geben Sie bitte die Daten einer Person ein: " );
  SIO.PutText("Name: ");
  resultat.name := SIO.GetLine();
  SIO.PutText("Vorname: ");
  resultat.vorname := SIO.GetLine();
  SIO.PutText("Alter: ");
  resultat.alter := SIO.GetInt();
  dummy := SIO.GetLine();  (* Lese RETURN aus Eingabe *)
  SIO.Nl();

  RETURN resultat;
END LesePerson;


PROCEDURE DruckePerson(person: Person)=
(* Gibt die Daten der Person person aus *)
BEGIN
  SIO.PutText(person.name & "  " & person.vorname & "  ");
  SIO.PutInt(person.alter);
  SIO.Nl();
END DruckePerson;


(* Implementierungen zum Prozedurtyp SelektProz *)

PROCEDURE Juenger18(person: Person): BOOLEAN=
(* TRUE, wenn die Person juenger als 18 ist, sonst FALSE. * )
BEGIN
  RETURN (person.alter < 18);
END Juenger18;


PROCEDURE VornameBertAlter20(person: Person): BOOLEAN=
(* TRUE, wenn die Person den Vornamen Bert hat und 20 ist, sonst FALSE. * )
BEGIN
  RETURN Text.Equal(person.vorname, "Bert") AND
         (person.alter = 20);
END VornameBertAlter20;
```

```
PROCEDURE NameA(person: Person): BOOLEAN=
(* TRUE, wenn der Name der Person mit A beginnt, sonst FALSE. *)

VAR zeichen: CHAR;

BEGIN
  zeichen := Text.GetChar(person.name, 0);
  RETURN (zeichen = 'A') OR (zeichen = 'a');
END NameA;


BEGIN
  (* Initialisiere Liste mit Personendaten *)
  InitListe(liste);
  FOR i := 1 TO 10 DO
    neuPerson := LesePerson();
    neuListeRef := ErzeugeElementListe(neuPerson);
    HaengeAnListe(liste, neuListeRef);
  END;

  (* Drucke urspruengliche Liste *)
  SIO.Nl();
  DruckeListe(liste);
  SIO.Nl();

  (* Bestimme Teilmengen und drucke das Ergebnis *)
  InitListe(ergListe);
  ergListe := IteriereListe(liste, Juenger18);
  SIO.Nl();
  SIO.PutLine("Alle Personen juenger 18: ");
  DruckeListe(ergListe);
  SIO.Nl();


  InitListe(ergListe);
  ergListe := IteriereListe(liste, VornameBertAlter20);
  SIO.Nl();
  SIO.PutLine("Alle Personen mit Vorname Bert und Alter 20: ");
  DruckeListe(ergListe);
  SIO.Nl();

  InitListe(ergListe);
  ergListe := IteriereListe(liste, NameA);
  SIO.Nl();
  SIO.PutLine("Alle Personen deren Name mit A beginnt: ");
  DruckeListe(ergListe);
  SIO.Nl();

END Datenbank121.
```

Ein Dialog mit dem Programm (**mit Benutzereingaben**) sieht dann folgendermaßen aus:

```
Geben Sie bitte die Daten einer Person ein:
Name: X.
Vorname: Heike
Alter: 32

Geben Sie bitte die Daten einer Person ein:
Name: W.
Vorname: Bert
Alter: 34
```

```
Geben Sie bitte die Daten einer Person ein:
Name: Z.
Vorname: Guido
Alter: 17

Geben Sie bitte die Daten einer Person ein:
Name: T.
Vorname: Ursula
Alter: 13

Geben Sie bitte die Daten einer Person ein:
Name: B.
Vorname: Bert
Alter: 20

Geben Sie bitte die Daten einer Person ein:
Name: A.
Vorname: Franka
Alter: 19

Geben Sie bitte die Daten einer Person ein:
Name: G.
Vorname: Karl
Alter: 50

Geben Sie bitte die Daten einer Person ein:
Name: A.
Vorname: Bertha
Alter: 15

Geben Sie bitte die Daten einer Person ein:
Name: F.
Vorname: Henry
Alter: 7

Geben Sie bitte die Daten einer Person ein:
Name: D.
Vorname: Doris
Alter: 32

X.  Heike  32
W.  Bert  34
Z.  Guido  17
T.  Ursula  13
B.  Bert  20
A.  Franka  19
G.  Karl  50
A.  Bertha  15
F.  Henry  7
D.  Doris  32

Alle Personen juenger 18:
Z.  Guido  17
T.  Ursula  13
A.  Bertha  15
F.  Henry  7

Alle Personen mit Vorname Bert und Alter 20:
B.  Bert  20

Alle Personen deren Name mit A beginnt:
A.  Franka  19
A.  Bertha  15
```

Lösung 12.2: ADT für Geldbeträge

Schnittstelle eines ADT für Währungsinformation zu einem Geldbetrag:

```
INTERFACE Waehrung;

(* Schnittstelle eines ADTs fuer Waehrungen.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 13.01.99  Letzte Aenderung: 13.01.99 *)


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

PROCEDURE ErzeugeCHF(): T;
(* Erzeugt ein Element des Typs fuer die Waehrung CHF. * )

PROCEDURE ErzeugeDM(): T;
(* Erzeugt ein Element des Typs fuer die Waehrung DM. * )

PROCEDURE ErzeugeEUR(): T;
(* Erzeugt ein Element des Typs fuer die Waehrung EUR. * )

PROCEDURE ErzeugeFF(): T;
(* Erzeugt ein Element des Typs fuer die Waehrung FF. *)

PROCEDURE ErzeugeGBP(): T;
(* Erzeugt ein Element des Typs fuer die Waehrung GBP. * )

PROCEDURE ErzeugeUSD(): T;
(* Erzeugt ein Element des Typs fuer die Waehrung USD. * )

PROCEDURE GibBezeichnung(waehrung: T ): TEXT;
(* Gibt die Bezeichnung der als Parameter gegebenen Waehrung zurueck. * )

PROCEDURE GibKurs(waehrung: T): REAL;
(* Gibt den Kurs der als Parameter gegebenen Waehrung zurueck. * )

END Waehrung.
```

Die Schnittstelle des ADTs für Geldbeträge:

```
INTERFACE Geldbetrag;

(* Schnittstelle eines ADTs fuer Geldbetraege.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 13.01.99  Letzte Aenderung: 13.01.99 *)

IMPORT Waehrung; (* Importiere ADT Waehrung *)

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

PROCEDURE Erzeuge(wert: REAL; waehrung: Waehrung.T): T;
(* Erzeugt einen neuen Geldbetrag mit dem gegebenen Wert in der
   angegebenen Waehrung. *)

PROCEDURE Tausche(betrag: T; waehrung: Waehrung.T): T;
(* Tauscht den gegebenen Geldbetrag in die angegebene Waehrung und
   gibt den Geldbetrag zurueck. *)

PROCEDURE Addiere(aBetrag, bBetrag: T): T;
(* Addiert den Geldetrag bBetrag zum Betrag aBetrag und gibt das
   Ergebnis in der Waehrung von aBetrag zurueck. *)

PROCEDURE Subtrahiere(aBetrag, bBetrag: T): T;
(* Subtrahiert den Geldbetrag bBetrag vom Betrag aBetrag und gibt
   das Ergebnis in der Waehrung von aBetrag zurueck. *)

PROCEDURE Multipliziere(betrag: T; skalar: REAL): T;
(* Multipliziert den Geldbetrag betrag mit dem gegebenen Skalarwert
   und gibt das Ergebnis in der Waehrung von betrag zurueck. *)

PROCEDURE GibText(betrag: T): TEXT;
(* Gibt den Wert des Geldbetrags betrag mit seiner Waehrungsbezeichnung
   als TEXT zurueck. *)

PROCEDURE Gleich(aBetrag, bBetrag: T): BOOLEAN;
(* Prueft, ob die beiden Geldbetraege gleich gross sind. *)

PROCEDURE Kleiner(aBetrag, bBetrag: T): BOOLEAN;
(* Prueft, ob der Geldbetrag aBetrag kleiner als bBetrag ist. *)

PROCEDURE Groesser(aBetrag, bBetrag: T): BOOLEAN;
(* Prueft, ob der Geldbetrag aBetrag groesser als bBetrag ist. *)

END Geldbetrag.
```

Implementierung zum ADT Waehrung:

```
MODULE Waehrung;

(* Implementiert den ADT fuer Waehrungen.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 13.01.99  Letzte Aenderung: 13.01.99 *)


TYPE REVEAL T = BRANDED REF RECORD
                  bezeichnung: TEXT;
                  kurs       : REAL;
                END;

PROCEDURE ErzeugeCHF(): T=
(* Erzeugt ein Element des Typs fuer die Waehrung CHF. * )
VAR neueWaehrung: T;

BEGIN
  neueWaehrung              := NEW(T);
  neueWaehrung^.bezeichnung := "CHF";
  neueWaehrung^.kurs        := 1.5883; (* 14.01.99 *)

  RETURN neueWaehrung;
END ErzeugeCHF;

PROCEDURE ErzeugeDM(): T=
(* Erzeugt ein Element des Typs fuer die Waehrung DM. * )
VAR neueWaehrung: T;

BEGIN
  neueWaehrung              := NEW(T);
  neueWaehrung^.bezeichnung := "DM";
  neueWaehrung^.kurs        := 1.95583;

  RETURN neueWaehrung;
END ErzeugeDM;

PROCEDURE ErzeugeEUR(): T=
(* Erzeugt eine Element des Typs fuer die Waehrung EUR. * )
VAR neueWaehrung: T;

BEGIN
  neueWaehrung              := NEW(T);
  neueWaehrung^.bezeichnung := "EUR";
  neueWaehrung^.kurs        := 1.00;

  RETURN neueWaehrung;
END ErzeugeEUR;

PROCEDURE ErzeugeFF(): T=
(* Erzeugt eine Element des Typs fuer die Waehrung FF. *)
VAR neueWaehrung: T;

BEGIN
  neueWaehrung              := NEW(T);
  neueWaehrung^.bezeichnung := "FF";
  neueWaehrung^.kurs        := 6.55957;

  RETURN neueWaehrung;
END ErzeugeFF;
```

```
PROCEDURE ErzeugeGBP(): T=
(* Erzeugt ein Element des Typs fuer die Waehrung GBP. * )
VAR neueWaehrung: T;

BEGIN
  neueWaehrung              := NEW(T);
  neueWaehrung^.bezeichnung := "GBP";
  neueWaehrung^.kurs        := 0.7067; (* 14.01.99 *)

  RETURN neueWaehrung;
END ErzeugeGBP;


PROCEDURE ErzeugeUSD(): T=
(* Erzeugt ein Element des Typs fuer die Waehrung USD. * )
VAR neueWaehrung: T;

BEGIN
  neueWaehrung              := NEW(T);
  neueWaehrung^.bezeichnung := "USD";
  neueWaehrung^.kurs        := 1.16615; (* 14.01.99 *)

  RETURN neueWaehrung;
END ErzeugeUSD;


PROCEDURE GibBezeichnung(waehrung: T ): TEXT=
(* Gibt die Bezeichnung der als Parameter gegebenen Waehrung zurueck. * )
BEGIN
  RETURN waehrung^.bezeichnung;
END GibBezeichnung;


PROCEDURE GibKurs(waehrung: T ): REAL=
(* Gibt den Kurs der als Parameter gegebenen Waehrung zurueck. * )
BEGIN
  RETURN waehrung^.kurs;
END GibKurs;

BEGIN
END Waehrung.
```

Implementierung zum ADT Geldbetrag:

```
MODULE Geldbetrag;

(* Implementiert den ADT fuer Geldbetraege.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 13.01.99  Letzte Aenderung: 13.01.99 *)


IMPORT Fmt;      (* Operationen zur Formatierung von INTEGER als TEXT. * )
IMPORT Waehrung; (* Importiere den ADT Waehrung *)

CONST Genauigkeit = 10000.00;  (* Speichere Betragswert auf hundertstel Cent *)


TYPE REVEAL T = BRANDED REF RECORD
                  wert       : INTEGER;   (* Wert entsprechend Genauigkeit *)
                  waehrung   : Waehrung.T;
                END;
```

```
PROCEDURE Erzeuge(wert: REAL; waehrung: Waehrung.T): T=
(* Erzeugt einen neuen Geldbetrag mit dem gegebenen Wert in der
   angegebenen Waehrung. *)

VAR neuerGeldbetrag: T;

BEGIN
  neuerGeldbetrag := NEW(T);
  neuerGeldbetrag^.waehrung := waehrung;
  neuerGeldbetrag^.wert := ROUND((wert * Genauigkeit) /
Waehrung.GibKurs(waehrung));

  RETURN neuerGeldbetrag;
END Erzeuge;


PROCEDURE Tausche(betrag: T; waehrung: Waehrung.T): T=
(* Tauscht den gegebenen Geldbetrag in die angegebene Waehrung und
   gibt den Geldbetrag zurueck. *)

VAR resGeldbetrag: T;

BEGIN
  resGeldbetrag           := NEW(T);
  resGeldbetrag^.wert     := betrag^.wert;
  resGeldbetrag^.waehrung := waehrung;

  RETURN resGeldbetrag;
END Tausche;


PROCEDURE Addiere(aBetrag, bBetrag: T): T=
(* Addiert den Geldetrag bBetrag zum Betrag aBetrag und gibt das
   Ergebnis in der Waehrung von aBetrag zurueck. *)

VAR resGeldbetrag: T;

BEGIN
  resGeldbetrag := NEW(T);
  resGeldbetrag^.wert := aBetrag^.wert + bBetrag^.wert;
  resGeldbetrag^.waehrung := aBetrag^.waehrung;

  RETURN resGeldbetrag;
END Addiere;


PROCEDURE Subtrahiere(aBetrag, bBetrag: T): T=
(* Subtrahiert den Geldbetrag bBetrag vom Betrag aBetrag und gibt
   das Ergebnis in der Waehrung von aBetrag zurueck. *)
VAR resGeldbetrag: T;

BEGIN
  resGeldbetrag := NEW(T);
  resGeldbetrag^.wert := aBetrag^.wert - bBetrag^.wert;
  resGeldbetrag^.waehrung := aBetrag^.waehrung;

  RETURN resGeldbetrag;
END Subtrahiere;
```

```
PROCEDURE Multipliziere(betrag: T; skalar: REAL): T=
(* Multipliziert den Geldbetrag betrag mit dem gegebenen Skalarwert
   und gibt das Ergebnis in der Waehrung von betrag zurueck. * )
VAR resGeldbetrag: T;

BEGIN
  resGeldbetrag := NEW(T);
  resGeldbetrag^.wert := ROUND(FLOAT(betrag^.wert, REAL) * skalar);
  resGeldbetrag^.waehrung := betrag^.waehrung;

  RETURN resGeldbetrag;
END Multipliziere;
PROCEDURE GibText(betrag: T): TEXT=
(* Gibt den Wert des Geldbetrags betrag mit seiner
   Waehrungsbezeichnung als TEXT zurueck. * )
VAR ausgabeWert  : REAL;
    vorKommaWert : REAL;
    nachKommaWert: REAL;
    resultat     : TEXT;

BEGIN
  ausgabeWert   := ( FLOAT(betrag^.wert, REAL) *
                     Waehrung.GibKurs(betrag^.waehrung));
  vorKommaWert  := ausgabeWert / Genauigkeit;
  nachKommaWert := ( vorKommaWert -
                     FLOAT(TRUNC(vorKommaWert), REAL)) * 100.00;
  resultat := Fmt.Int(TRUNC(vorKommaWert), 10);

  IF (ABS(ROUND(nachKommaWert)) < 10) THEN
    (* Fuehrende Null fuer Nachkommawerte kleiner 10 *)
    resultat := resultat & ".0";
  ELSE
    resultat := resultat & ".";
  END;

  resultat := resultat & Fmt.Int(ABS(ROUND(nachKommaWert)));
  resultat := resultat & " " & Waehrung.GibBezeichnung(betrag^.waehrung);

  RETURN resultat;
END GibText;


PROCEDURE Gleich(aBetrag, bBetrag: T): BOOLEAN=
(* Prueft, ob die beiden Geldbetraege gleich gross sind. * )
BEGIN
  RETURN aBetrag^.wert = bBetrag^.wert;
END Gleich;


PROCEDURE Kleiner(aBetrag, bBetrag: T): BOOLEAN=
(* Prueft, ob der Geldbetrag aBetrag kleiner als bBetrag ist. * )
BEGIN
  RETURN aBetrag^.wert < bBetrag^.wert;
END Kleiner;


PROCEDURE Groesser(aBetrag, bBetrag: T): BOOLEAN=
(* Prueft, ob der Geldbetrag aBetrag groesser als bBetrag ist. * )
BEGIN
  RETURN aBetrag^.wert > bBetrag^.wert;
END Groesser;

BEGIN
END Geldbetrag.
```

Das Hauptprogramm für den Test des ADT Geldbetrag:

```
MODULE Geldbetraege122 EXPORTS Main;

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

IMPORT Waehrung;   (* ADT fuer Waehrungsinformation. *)
IMPORT Geldbetrag; (* ADT fuer Geldbetraege. *)

VAR chfBetrag, dmBetrag, ffBetrag, eurBetrag:  Geldbetrag.T;
    gbpBetrag, usdBetrag, betrag            : Geldbetrag.T;

BEGIN
  (* Erzeuge einige Betraege in verschiedenen Waehrungen *)
  chfBetrag := Geldbetrag.Erzeuge(5.67, Waehrung.ErzeugeCHF());
  SIO.PutLine(Geldbetrag.GibText(chfBetrag));
  dmBetrag  := Geldbetrag.Erzeuge(17.34, Waehrung.ErzeugeDM());
  SIO.PutLine(Geldbetrag.GibText(dmBetrag));
  ffBetrag  := Geldbetrag.Erzeuge(345.90, Waehrung.ErzeugeFF());
  SIO.PutLine(Geldbetrag.GibText(ffBetrag));
  eurBetrag := Geldbetrag.Erzeuge(234.21, Waehrung.ErzeugeEUR());
  SIO.PutLine(Geldbetrag.GibText(eurBetrag));
  gbpBetrag := Geldbetrag.Erzeuge(9.16, Waehrung.ErzeugeGBP());
  SIO.PutLine(Geldbetrag.GibText(gbpBetrag));
  usdBetrag := Geldbetrag.Erzeuge(45.93, Waehrung.ErzeugeUSD());
  SIO.PutLine(Geldbetrag.GibText(usdBetrag));
  SIO.Nl();

  (* ... und fuehre Operationen darauf aus. *)
  betrag := Geldbetrag.Addiere(chfBetrag, dmBetrag);
  SIO.PutLine("Addition         CHF/DM : " & Geldbetrag.GibText(betrag));
  betrag := Geldbetrag.Subtrahiere(ffBetrag, eurBetrag);
  SIO.PutLine("Subtraktion      FF/EUR : " & Geldbetrag.GibText(betrag));
  betrag := Geldbetrag.Addiere(betrag, chfBetrag);
  SIO.PutLine("Addition   Ergebnis/CHF : " & Geldbetrag.GibText(betrag));
  betrag := Geldbetrag.Subtrahiere(usdBetrag, gbpBetrag);
  SIO.PutLine("Subtraktion     USD/GBP : " & Geldbetrag.GibText(betrag));
  betrag := Geldbetrag.Multipliziere(gbpBetrag, 1.18);
  SIO.PutLine("18 % Mehrwertsteuer GBP : " & Geldbetrag.GibText(betrag));
  betrag := Geldbetrag.Multipliziere(eurBetrag, 0.95);
  SIO.PutLine("5 % Rabatt          EUR : " & Geldbetrag.GibText(betrag));
  SIO.Nl();

  betrag := Geldbetrag.Tausche(usdBetrag, Waehrung.ErzeugeDM());
  SIO.PutLine("Tausche USD in DM  : " & Geldbetrag.GibText(betrag));
  SIO.PutText("USD-Betrag gleich Tausch? ");
  SIO.PutBool(Geldbetrag.Gleich(usdBetrag,  betrag));
  SIO.Nl();
  betrag := Geldbetrag.Tausche(betrag, Waehrung.ErzeugeUSD());
  SIO.PutLine("Tausche wieder USD : " & Geldbetrag.GibText(betrag));
  SIO.Nl();

  SIO.PutText("FF-  gleich   CHF-Betrag? ");
  SIO.PutBool(Geldbetrag.Gleich(ffBetrag, chfBetrag));
  SIO.Nl();
  SIO.PutText("GBP- kleiner  USD-Betrag? ");
  SIO.PutBool(Geldbetrag.Kleiner(gbpBetrag, usdBetrag));
  SIO.Nl();
  SIO.PutText("GBP- groesser USD-Betrag? ");
  SIO.PutBool(Geldbetrag.Groesser(gbpBetrag, usdBetrag));
  SIO.Nl();
  SIO.PutText("DM-  groesser CHF-Betrag? ");
  SIO.PutBool(Geldbetrag.Groesser(dmBetrag, chfBetrag));
  SIO.Nl();
```

```
  SIO.PutText("DM-  kleiner  CHF-Betrag? ");
  SIO.PutBool(Geldbetrag.Kleiner(dmBetrag, chfBetrag));
  SIO.Nl();

END Geldbetraege122.
```

Die Ausgabe des Programms sieht folgendermaßen aus:

```
5.67 CHF
17.34 DM
345.90 FF
234.21 EUR
9.16 GBP
45.93 USD

Addition          CHF/DM : 19.75 CHF
Subtraktion       FF/EUR : -1190.42 FF
Addition   Ergebnis/CHF : -1167.00 FF
Subtraktion      USD/GBP : 30.81 USD
18 % Mehrwertsteuer GBP : 10.81 GBP
5 % Rabatt           EUR : 222.50 EUR

Tausche USD in DM  : 77.03 DM
USD-Betrag gleich Tausch? TRUE
Tausche wieder USD : 45.93 USD

FF-  gleich   CHF-Betrag? FALSE
GBP- kleiner  USD-Betrag? TRUE
GBP- groesser USD-Betrag? FALSE
DM-  groesser CHF-Betrag? TRUE
DM-  kleiner  CHF-Betrag? FALSE
```

Lösungen zu Übung 13

Lösung 13.1: Einmaleins-Taschenrechner

Schnittstelle des Taschenrechner-Moduls:

```
INTERFACE Taschenrechner;

(* Schnittstelle des Objektmoduls fuer einen Einmaleins-Taschenrechner.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 21.01.99  Letzte Aenderung: 21.01.99 *)


EXCEPTION DivisionByZero;
          InvalidOperand;
          Overflow;
          Underflow;


PROCEDURE LeseOperand(): INTEGER RAISES {InvalidOperand};
(* Liest einen Operanden von der Standardeingabe. * )


PROCEDURE Addiere(operand1, operand2: INTEGER ): INTEGER
  RAISES {Overflow, InvalidOperand};
(* Addiert die zwei Werte operand1 und operand2 *)


PROCEDURE Subtrahiere(operand1, operand2: INTEGER ): INTEGER
  RAISES {Underflow, InvalidOperand};
(* Subtrahiert den Wert operand2 von operand1 *)


PROCEDURE Multipliziere(operand1, operand2: INTEGER ): INTEGER
  RAISES {Overflow, InvalidOperand};
(* Multipliziert die zwei Werte operand1 und operand2 *)


PROCEDURE Dividiere(operand1, operand2: INTEGER ): INTEGER
  RAISES {DivisionByZero, InvalidOperand};
(* Dividiert ganzzahlig die zwei Werte operand1 und operand2 *)


PROCEDURE BereichGueltig(a: INTEGER ): BOOLEAN;
(* Prueft, ob die Zahl im erlaubten Zahlenbereich des Taschenrechners liegt. * )


END Taschenrechner.
```

Implementierung des Taschenrechner-Moduls:

```
MODULE Taschenrechner;

(* Implementierung des Objektmoduls fuer einen Einmaleins-Taschenrechner.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 21.01.99  Letzte Aenderung: 21.01.99 *)

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

CONST MinZahl = 0;     (* Grenzen des gueltigen *)
      MaxZahl = 100;   (* Zahlenbereichs        *)


PROCEDURE LeseOperand(): INTEGER RAISES {InvalidOperand}=
(* Liest einen Operanden von der Standardeingabe. * )

VAR resultat: INTEGER;

BEGIN
  TRY
    resultat := SIO.GetInt();

    IF NOT BereichGueltig(resultat) THEN
      RAISE InvalidOperand; (* Operand ausserhalb Gueltigkeitsbereich *)
    END;
  EXCEPT
    (* Setze SIO.Error-Ausnahme in InvalidOperand-Ausnahme um *)
    SIO.Error => RAISE InvalidOperand;
  END;

  RETURN resultat;
END LeseOperand;

PROCEDURE Addiere(operand1, operand2: INTEGER): INTEGER
  RAISES {Overflow, InvalidOperand}=
(* Addiert die zwei Werte operand1 und operand2 *)
BEGIN
  IF NOT BereichGueltig(operand1) OR NOT BereichGueltig(operand2) THEN
    RAISE InvalidOperand; (* Operand ausserhalb Gueltigkeitsbereich *)
  END;

  IF NOT BereichGueltig(operand1 + operand2) THEN
    RAISE Overflow; (* Ergebnis groesser als Gueltigkeitsbereich *)
  END;

  RETURN operand1 + operand2;
END Addiere;

PROCEDURE Subtrahiere(operand1, operand2: INTEGER): INTEGER
  RAISES {Underflow, InvalidOperand}=
(* Subtrahiert den Wert operand2 von operand1 *)
BEGIN
  IF NOT BereichGueltig(operand1) OR NOT BereichGueltig(operand2) THEN
    RAISE InvalidOperand; (* Operand ausserhalb Gueltigkeitsbereich *)
  END;

  IF NOT BereichGueltig(operand1 - operand2) THEN
    RAISE Underflow; (* Ergebnis kleiner als Gueltigkeitsbereich *)
  END;

  RETURN operand1 - operand2;
END Subtrahiere;
```

```
PROCEDURE Multipliziere(operand1, operand2: INTEGER): INTEGER
  RAISES {Overflow, InvalidOperand}=
(* Multipliziert die zwei Werte operand1 und operand2 *)
BEGIN
  IF NOT BereichGueltig(operand1) OR NOT BereichGueltig(operand2) THEN
    RAISE InvalidOperand; (* Operand ausserhalb Gueltigkeitsbereich *)
  END;

  IF NOT BereichGueltig(operand1 * operand2) THEN
    RAISE Overflow; (* Ergebnis groesser als Gueltigkeitsbereich *)
  END;

  RETURN operand1 * operand2;
END Multipliziere;


PROCEDURE Dividiere(operand1, operand2: INTEGER ): INTEGER
  RAISES {DivisionByZero, InvalidOperand}=
(* Dividiert ganzzahlig die zwei Werte operand1 und operand2 *)
BEGIN
  IF NOT BereichGueltig(operand1) OR NOT BereichGueltig(operand2) THEN
    RAISE InvalidOperand; (* Operand ausserhalb Gueltigkeitsbereich *)
  END;

  IF (operand2 = 0) THEN
    RAISE DivisionByZero; (* Division durch Null *)
  END;

  RETURN operand1 DIV operand2;
END Dividiere;


PROCEDURE BereichGueltig(a: INTEGER): BOOLEAN=
(* Prueft, ob die Zahl im erlaubten Zahlenbereich des Taschenrechners liegt. * )
BEGIN
  RETURN (a >= MinZahl) AND (a <= MaxZahl);
END BereichGueltig;


BEGIN
END Taschenrechner.
```

Hauptprogramm, das die Berechnung einfacher Ausdrücke mit Hilfe des Taschenrechner-Moduls erlaubt und dabei auftretende Ausnahmen behandelt:

```
MODULE Einmaleins131 EXPORTS Main;

IMPORT SIO;               (* Importiere Ein-/Ausgabeoperationen *)
IMPORT Taschenrechner;    (* Importiere Modul fuer den Taschenrechner *)

<* FATAL SIO.Error *> (* Bricht Programm ab, falls SIO-Operation bei
                         Ausnahmebehandlung fehlschlaegt. * )

VAR operand1, operand2: INTEGER;
    operator          : CHAR;
    dummyLine         : TEXT;

BEGIN
  LOOP
    TRY
      SIO.PutLine("Was willst Du berechnen? ( z.B. 5+6 ) ");
      operand1  := Taschenrechner.LeseOperand();
      operator  := SIO.GetChar();
      operand2  := Taschenrechner.LeseOperand();
      dummyLine := SIO.GetLine();

      CASE operator OF
      | '+' => operand2 := Taschenrechner.Addiere(operand1, operand2);
      | '-' => operand2 := Taschenrechner.Subtrahiere(operand1, operand2);
      | '*' => operand2 := Taschenrechner.Multipliziere(operand1, operand2);
      | '/' => operand2 := Taschenrechner.Dividiere(operand1, operand2);
      ELSE
        RAISE SIO.Error; (* Falls unbekanntes Zeichen als Operator vorliegt *)
      END;

      (* Gib das Ergebnis aus *)
      SIO.PutInt(operand2); SIO.Nl(); SIO.Nl();

    EXCEPT
    | Taschenrechner.InvalidOperand =>
        SIO.PutLine("FEHLER: Ungueltige Zahl eingegeben!");
        dummyLine := SIO.GetLine(); (* Lese verbleibende Zeile *)

    | Taschenrechner.Overflow       =>
        SIO.PutLine("FEHLER: Gueltigen Bereich ueberschritten!");

    | Taschenrechner.Underflow      =>
        SIO.PutLine("FEHLER: Gueltigen Bereich unterschritten!");

    | Taschenrechner.DivisionByZero =>
        SIO.PutLine("FEHLER: Division durch Null!");

    | SIO.Error                     =>
        SIO.PutLine("FEHLER: Fehler bei der Eingabe!");
    END;
  END; (* LOOP *)

END Einmaleins131.
```

Ein Dialog mit dem Taschenrechner-Programm sieht folgendermaßen aus:

```
Was willst Du berechnen? ( z.B. 5+6 )
5+6
11

Was willst Du berechnen? ( z.B. 5+6 )
4*7
28

Was willst Du berechnen? ( z.B. 5+6 )
12/4
3

Was willst Du berechnen? ( z.B. 5+6 )
7-4
3

Was willst Du berechnen? ( z.B. 5+6 )
95+8
FEHLER: Gueltigen Bereich ueberschritten!

Was willst Du berechnen? ( z.B. 5+6 )
3-5
FEHLER: Gueltigen Bereich unterschritten!

Was willst Du berechnen? ( z.B. 5+6 )
7/0
FEHLER: Division durch Null!

Was willst Du berechnen? ( z.B. 5+6 )
x+6
FEHLER: Ungueltige Zahl eingegeben!

Was willst Du berechnen? ( z.B. 5+6 )
7k8
FEHLER: Fehler bei der Eingabe!

Was willst Du berechnen? ( z.B. 5+6 )
6 + 9
FEHLER: Ungueltige Zahl eingegeben!

Was willst Du berechnen? ( z.B. 5+6 )
6+9
15
```

Lösung 13.2: Getränkeautomat
Mögliche Schnittstelle des Getränkeautomaten:

```
INTERFACE Automat;

(* Schnittstelle eines Objektmoduls für einen Getraenkeautomaten.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 21.01.99  Letzte Aenderung: 21.01.99 *)


IMPORT Assertion;


PROCEDURE MuenzeEinwerfen() RAISES {Assertion.Violated};
(* Einwerfen einer Muenze in den Automaten. *)

PROCEDURE GeldZurueck() RAISES {Assertion.Violated};
(* Gibt eine eventuell eingeworfene Muenze zurueck. *)

PROCEDURE Auffuellen() RAISES {Assertion.Violated};
(* Fuelle ein Getraenk in den Automaten ein. *)

PROCEDURE Entnehmen() RAISES {Assertion.Violated};
(* Entnehme bezahltes Getraenk aus dem Automaten. *)

PROCEDURE MuenzeEingeworfen(): BOOLEAN;
(* Prueft, ob Muenze eingeworfen wurde. *)

PROCEDURE Voll(): BOOLEAN;
(* Prueft, ob der Automat voll ist. *)

PROCEDURE Leer(): BOOLEAN;
(* Prueft, ob der Automat leer ist. *)

END Automat.
```

Implementierung des Getränkeautomaten:

```
MODULE Automat;

(* Implementierung des Objektmoduls für einen Getraenkeautomaten.
   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : SRC-Modula-3 rel. 3.6, Windows NT 4.0
   Erstellt        : 21.01.99  Letzte Aenderung: 21.01.99 *)

IMPORT Assertion; (* Importiere das Assertion-Modul *)

CONST MaxGetraenke = 5; (* Maximale Anzahl Getraenke im Automaten *)

VAR getraenke   : [0 .. MaxGetraenke];
    mEingeworfen: BOOLEAN;


PROCEDURE MuenzeEinwerfen() RAISES {Assertion.Violated}=
(* Einwerfen einer Muenze in den Automaten. *)
BEGIN
  Assertion.Require(NOT MuenzeEingeworfen(), "MuenzeEinwerfen");
  mEingeworfen := TRUE;
  Assertion.Ensure(MuenzeEingeworfen(), "MuenzeEinwerfen");
END MuenzeEinwerfen;
```

```
PROCEDURE GeldZurueck() RAISES {Assertion.Violated}=
(* Gibt eine eventuell eingeworfene Muenze zurueck. *)
BEGIN
  Assertion.Require(MuenzeEingeworfen(), "GeldZurueck");
  mEingeworfen := FALSE;
  Assertion.Ensure(NOT MuenzeEingeworfen(), "GeldZurueck");
END GeldZurueck;


PROCEDURE Auffuellen() RAISES {Assertion.Violated}=
(* Fuelle ein Getraenk in den Automaten ein. *)
BEGIN
  Assertion.Require(NOT Voll(), "Auffuellen");
  getraenke := getraenke + 1;
  Assertion.Ensure(NOT Leer(), "Auffuellen");
END Auffuellen;


PROCEDURE Entnehmen() RAISES {Assertion.Violated}=
(* Entnehme bezahltes Getraenk aus dem Automaten. *)
BEGIN
  Assertion.Require(MuenzeEingeworfen() AND NOT Leer(), "Entnehmen");
  getraenke := getraenke - 1;
  mEingeworfen := FALSE;
  Assertion.Ensure(NOT MuenzeEingeworfen() AND NOT Voll(), "Entnehmen");
END Entnehmen;


PROCEDURE MuenzeEingeworfen(): BOOLEAN=
(* Prueft, ob Muenze eingeworfen wurde. *)
BEGIN
  RETURN mEingeworfen;
END MuenzeEingeworfen;


PROCEDURE Voll(): BOOLEAN=
(* Prueft, ob der Automat voll ist. *)
BEGIN
  RETURN (getraenke = MaxGetraenke);
END Voll;


PROCEDURE Leer(): BOOLEAN=
(* Prueft, ob der Automat leer ist. *)
BEGIN
  RETURN (getraenke = 0);
END Leer;


BEGIN
  (* Initialisiere den Automaten als leer. *)
  getraenke := 0;
  mEingeworfen := FALSE;
END Automat.
```

Hauptprogramm für die Erprobung des Getränkeautomaten-Objektmoduls:

```
MODULE GetraenkeAutomat132 EXPORTS Main;

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

IMPORT Assertion;  (* Operationen fuer Zusicherungen. *)
IMPORT Automat;    (* Objektmodul fuer den Getraenkeautomaten. *)

BEGIN
  (* Assertion.DisableAssertions(); *) (* Zum Vergleich! *)

  TRY
    Automat.Auffuellen();
    Automat.Auffuellen();
    Automat.Auffuellen();
    Automat.Auffuellen();
    Automat.Auffuellen();

    SIO.PutLine("1. Automat mit 5 Getraenken aufgefuellt!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("1. Auffuellen nicht moeglich!");
                          SIO.Nl();
  END;

  TRY
    Automat.Auffuellen();
    SIO.PutLine("2. Automat mit weiterem Getraenk gefuellt!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("2. Auffuellen nicht moeglich!");
                          SIO.Nl();
  END;

  TRY
    Automat.Entnehmen();
    SIO.PutLine("3. Getraenk entnommen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("3. Entnehmen nicht moeglich!");
                          SIO.Nl();
  END;

  TRY
    Automat.MuenzeEinwerfen();
    SIO.PutLine("4. Muenze eingeworfen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("4. Einwerfen nicht moeglich!");
  END;

  TRY
    Automat.MuenzeEinwerfen();
    SIO.PutLine("5. Muenze eingeworfen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("5. Einwerfen nicht moeglich!");
                          SIO.Nl();
  END;
```

```
  TRY
    Automat.Entnehmen();
    SIO.PutLine("6. Getraenk entnommen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("6. Entnehmen nicht moeglich!");
                          SIO.Nl();
  END;

  TRY
    Automat.MuenzeEinwerfen();
    Automat.Entnehmen();
    Automat.MuenzeEinwerfen();
    Automat.Entnehmen();
    Automat.MuenzeEinwerfen();
    Automat.Entnehmen();
    Automat.MuenzeEinwerfen();
    Automat.Entnehmen();
    SIO.PutLine("7. Vier weitere Getraenke entnommen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("7. Fehler bei Entnahme vier weiterer
Getraenke!");
                          SIO.Nl();
  END;

  TRY
    Automat.MuenzeEinwerfen();
    Automat.Entnehmen();
    SIO.PutLine("8. Noch ein Getraenk entnommen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("8. Entnehmen nicht moeglich!");
                          SIO.Nl();
  END;


  TRY
    Automat.GeldZurueck();
    SIO.PutLine("9. Geld zurueckbekommen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("9. Geld zurueck nicht moeglich!");
                          SIO.Nl();
  END;

  TRY
    Automat.GeldZurueck();
    SIO.PutLine("10. Geld zurueckbekommen!");
    SIO.Nl();
  EXCEPT
    Assertion.Violated => SIO.PutLine("10. Geld zurueck nicht moeglich!");
                          SIO.Nl();
  END;

END GetraenkeAutomat132.
```

Eine Ausführung des Programms produziert folgende Ausgabe:

```
1. Automat mit 5 Getraenken aufgefuellt!

*** Precondition violated: Auffuellen
2. Auffuellen nicht moeglich!

*** Precondition violated: Entnehmen
3. Entnehmen nicht moeglich!

4. Muenze eingeworfen!

*** Precondition violated: MuenzeEinwerfen
5. Einwerfen nicht moeglich!

6. Getraenk entnommen!

7. Vier weitere Getraenke entnommen!

*** Precondition violated: Entnehmen
8. Entnehmen nicht moeglich!

9. Geld zurueckbekommen!

*** Precondition violated: GeldZurueck
10. Geld zurueck nicht moeglich!
```