Programmierung	WS 2000/01	**Prof. H. Lichter** **A. Nowack**

Musterlösung
Testklausur

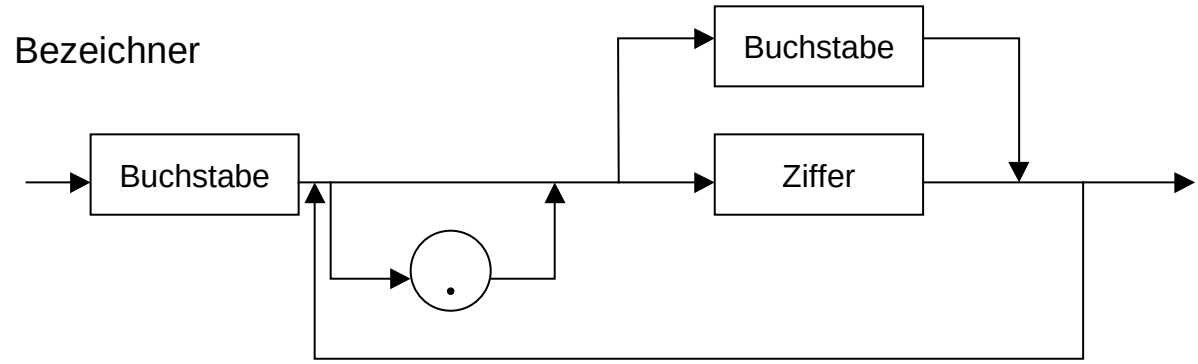
Aufgabe 1 : Syntaxdiagramm und EBNF

Es sollen Bezeichner definiert werden, die folgenden Vorschriften genügen:

- Bezeichner bestehen aus Buchstaben, Ziffern und Punkten.
- Jeder Bezeichner beginnt mit einem Buchstaben.
- Die Bezeichner sind beliebig lang, aber nicht kürzer als zwei Zeichen
- Bezeichner dürfen durch Punkte gegliedert werden, jedoch dürfen diese weder am Ende noch mehrfach hintereinander stehen (d.h. „ABC.“ Und „AB..C“ sind falsch).

Die Produktionen für **Buchstabe** und **Ziffer** können als gegeben betrachtet werden.

1.1 Geben Sie die Syntax für Bezeichner durch ein Syntaxdiagramm an!
Die Lösung muß knapp und eindeutig sein.

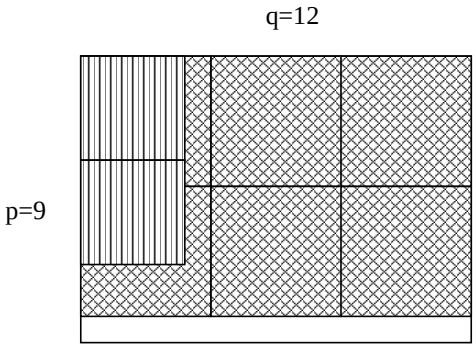


1.2 Wie verändert sich die Lösung, wenn zusätzlich gelten soll, daß ein Bezeichner höchstens eine Ziffer enthalten darf?
Formulieren Sie die veränderte Syntaxbeschreibung in EBNF!

```
Bezeichner      = Buchstabe Rest
Rest            = ["."] ( Buchstabe  [Rest] | Ziffer { ["." ] Buchstabe } )
```

Aufgabe 2: Kacheln

Bei der folgenden Aufgabe sind alle Längenmaße Vielfache der Grundeinheit E = 10 cm. Die Gewichte (präzise: Massen) sind Vielfache der Masse einer kleinen Kachel (10 cm im Quadrat), die m = 100 g wiegt. Die Kachel mit doppelter Kantenlänge ist entsprechend viermal so schwer usw.
Es stehen quadratische Kacheln aller Größenstufen bis zur Kantenlänge kmax*E zur Verfügung, also mit den Kantenlängen E, 2E, ... , kmax*E.
Ein Rechteck der Länge p*E und der Breite q*E (p, q $\in$ N) ist mit den größten verfügbaren und in das Rechteck passenden Kacheln zu belegen, wobei diese nicht über das Rechteck hinausragen dürfen. Sie bilden nun ein neues Rechteck, das mit den nächstkleineren Kacheln ausgelegt wird usw., bis schließlich in der obersten Schicht Kacheln der Größe E liegen.
Für ein gegebenes Tripel p, q, kmax ist das Gesamtgewicht der Kacheln zu berechnen, die auf dem Rechteck liegen. Beispiel: p = 9, q = 12, kmax = 4.



Die Abbildung zeigt, daß 2*3=6 Kacheln der Größe 4 auf das Rechteck passen, darauf 2*4=8 Kacheln der Größe 3 (von denen zwei links gezeigt sind). Auf diese wiederum kommen 3*6 der Größe 2 und 6*12 der Größe 1. Das ergibt ein Gesamtgewicht von (6*16 + 8*9 + 18*4 + 72*1) * 100 g = 31 200 g.

2.1 Geben Sie eine Modula-3 Funktion an, die aus den Parametern p, q, und kmax das Gewicht (in g) der Kacheln nach obigem Berechnungsschema liefert.

```
PROCEDURE BerechneKacheln (p, q, kmax : INTEGER): INTEGER =
VAR panz, qanz : INTEGER;
BEGIN
  IF kmax > 0 THEN
    qanz := q DIV kmax; panz := p DIV kmax;

    IF (panz > 0) AND (qanz > 0) THEN

      (* Kacheln der Breite bzw. Höhe kmax können verwendet werden *)
      SIO.PutInt((qanz * panz) * (kmax *kmax)); SIO.Nl();

      (* Masse dieser Schicht + Masse des rekursiv berechneten Rests = Ergebnis *)
      RETURN (((qanz * panz) * (kmax *kmax) * 100) +
                  BerechneKacheln(panz * kmax, qanz * kmax, kmax - 1));
    ELSE
      (* Probiere nächst kleinere Kacheln! *)
      RETURN BerechneKacheln (p, q, kmax - 1);
    END;
  ELSE
    RETURN 0;
  END;
END BerechneKacheln;
```

Aufgabe 3: Programmanalyse

Gegeben sei die folgende rekursive Prozedur:

```
PROCEDURE Unknown (p1 : INTEGER; p2 : BOOLEAN) =
BEGIN
  IF NOT p2 THEN
    SIO.PutInt (p1 DIV 7);
    SIO.Nl();
  END;
  IF p2 OR (p1 > 10) THEN
    Unknown ( (p1 + 33) MOD 41, p2 # (p1 < 20))
  END;
END Unknown;
```

Die Prozedur erzeugt also mit jeder Inkarnation keine oder ein Zeile Augabe.

3.1 Geben Sie nach dem Muster des Beispiels (Aufruf Unknown (10, TRUE)) die Inkarnationen von Unknown an, die aus dem Aufruf Unknown (30, FALSE) entstehen, und zwar bis zum Abbruch der Rekursion oder bis zur letzten Zeile in der dafür vorgesehenen Tabelle.

#	p1	p2	Ausgabe (falls erzeugt)
1	10	TRUE	-
2	2	FALSE	0 Abbruch

#	p1	p2	Ausgabe (falls erzeugt)
1	30	FALSE	4
2	22	FALSE	3
3	14	FALSE	2
4	6	TRUE	-
5	39	FALSE	5
6	31	FALSE	4
7	23	FALSE	3

Aufgabe 4: Listen

Seien $x = (x_1,x_2,\ldots,x_m)$ und $y = (y_1,y_2, \ldots, y_n)$ zwei nicht-leere einfach verkettete Listen.

Aus diesen Listen soll kann eine neue Liste Z konstruiert werden, die folgenden Bedingungen genügt.

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

4.1 Entwerfen Sie eine geeignete Datenstruktur, und schreiben Sie eine Modula-3 Prozedur `ErzeugeZ`, die aus zwei nicht-leeren einfach verketteten Listen x und y eine neue Liste z gemäß obiger Spezifikation erzeugt (die beiden Eingabe-Listen werden dabei zerstört)

Datenstruktur:

```
TYPE Liste = REF ListenElement;
     ListenElement = RECORD
                       inhalt : INTEGER;
                       naechster : Liste;
                     END;
```

Prozedur:

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

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

 (* Durchlaufen der Listen, bis das Ende einer Liste erreicht ist *)
  WHILE x # NIL AND y # NIL DO
    hz^.nachfolger := x;
    x := x^.nachfolger;
    hz := hz^.nachfolger;
    hz^.nachfolger := y;
    y := y^.nachfolger;
    hz := hz^.nachfolger;
  END;

  (* Hänge Rest der Liste an, deren Ende noch nicht erreicht wurde *)
  IF x = NIL THEN
   hz^.nachfolger := y;
  ELSE
   hz^.nachfolger := x;
  END;
END MischeListen;
```

Die beiden Listen werden (am Anfang beginnend) durchlaufen, bis bei einer der beiden Listen das Ende erreicht ist. Während des Durchlaufens werden abwechselnd Elemente aus den beiden Listen in die Ergebnisliste eingefügt. Ist das Ende der einen Liste erreicht, das Ende der anderen jedoch noch nicht, so wird der Rest der Liste, deren Ende noch nicht erreicht ist, an das Ende der Ergebnisliste angehängt.

Aufgabe 5: Abstrakter Datentyp

Die Realisierung von Mengen in Modula-3 ist beschränkt auf Ordinaltypen als Elementtyp der Menge. Sollen andere Elementtypen in Mengen verwendet werden, so muß dafür eine geeignete Implementierung erstellt werden.

Entwerfen Sie einen Abstrakten Datentyp SetOfText, der eine Menge realisiert, deren Elementtyp der vordefinierte Typ TEXT ist. Als Operationen sollen bereitgestellt werden: Vereinigung, Schnitt, Aufnahme und Entfernen eines Elements aus der Menge sowie der Test des Enthaltenseins.

5.1 Geben Sie das Interface-Modul für den ADT SetOfText an!

```
INTERFACE SetOfTextADT;

TYPE SetOfText <: REFANY;

PROCEDURE Erzeuge () : SetOfText;
PROCEDURE Vereinigung (s1,s2 : SetOfText) : SetOfText;
PROCEDURE Schnitt (s1,s2 : SetOfText) : SetOfText;
PROCEDURE Aufnehmen (VAR s : SetOfText; t : TEXT);
PROCEDURE Entfernen (VAR s : SetOfText; t : TEXT);
PROCEDURE Enthaelt (s: SetOfText; t : TEXT) : BOOLEAN;


END SetOfTextADT.
```

5.2 Geben Sie den Deklarationsteil des Implementierungs-Moduls des ADTs `SetOfText` an (d.h. alles bis zur ersten implementierten Funktion).
Dabei sollen Mengen prinzipiell unendlich viele Elemente aufnehmen können.

```
REVEAL SetOfText = BRANDED REF RECORD
                     t : TEXT;
                     naechstesElement : SetOfText;
                END;
```

5.3 Implementieren Sie die Funktion der Mengenvereinigung des ADTs `SetOfText` gemäß Ihrer Deklaration im Interface-Modul!

Die angegebene Prozedur arbeitet wie folgt:
Die “Menge“ resSet soll am Ende das Ergebnis der Vereinigung enthalten.
Zunächst wird resSet mit der “leeren Menge“ initialisiert.
Dann werden nacheinander alle Elemente von s1 in die “Menge“ resSet eingefügt, danach diejenigen aus s2.
Das Einfügen geschieht mit Hilfe der Prozedur “Aufnehmen“. Bei der Verwendung von “Aufnehmen“ wird wird vorausgesetzt, daß ein Element, das in der “Menge“ bereits vorhanden ist, nicht noch einmal eingefügt wird. Somit müssen Elemente, die in beiden Mengen auftreten, nicht durch einen Sonderfall behandelt werden.

```
PROCEDURE Vereinigung (s1, s2 : SetOfText) : SetOfText =
VAR resSet : SetOfText;

BEGIN
  (* Initialisieren mit der leeren Menge *)
  resSet := Erzeuge();

  (* Einfügen der Element von s1 *)
  WHILE s1 # NIL DO
    Aufnehmen(resSet, s1.t);
    s1 := s1^.naechstesElement;
  END;

  (* Einfügen der Elemente von s2 *)
  WHILE s2 # NIL DO
    Aufnehmen(resSet, s2.t);
    s2 := s2^.naechstesElement;
  END;
  RETURN resSet;
END Vereinigung;
```

Aufgabe 6: Klassenhierarchie und Objekttypen

Sie haben die Aufgabe, ein Programm zu entwickeln, mit dem Literaturreferenzen verwaltet werden sollen. Dabei stellen Sie fest, daß es die folgenden vier verschiedenen Arten von Literaturreferenzen gibt.

Referenz auf ein Buch : diese ist charakterisiert durch Angabe von Autor, Titel, Erscheinungsjahr und Verlag

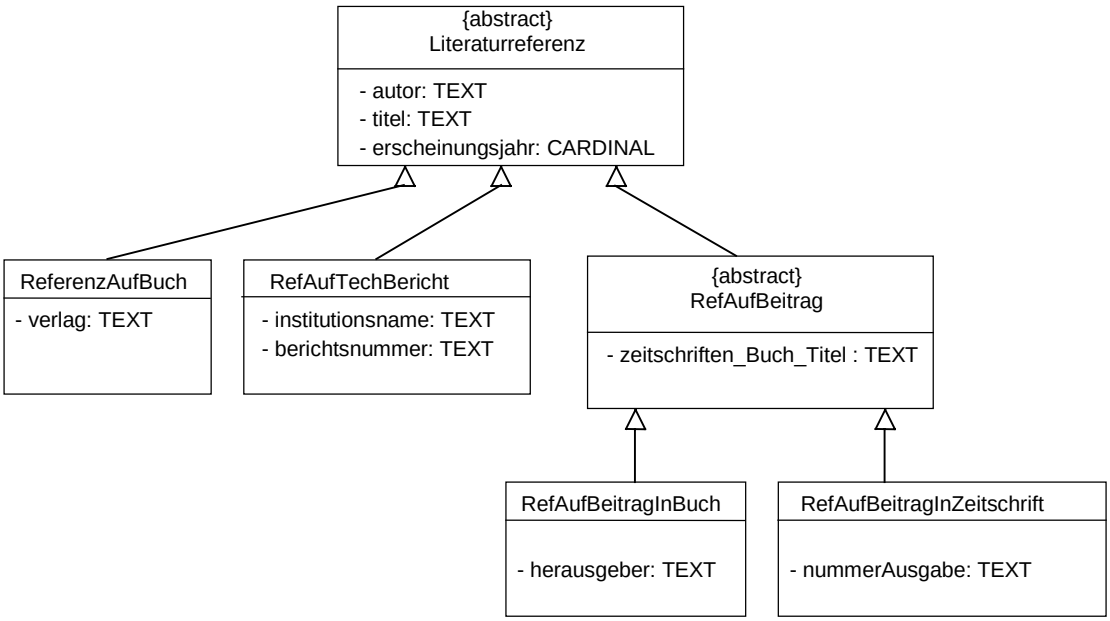
Referenz auf einen Technischen Bericht: diese ist charakterisiert durch Angabe von Autor, Titel, Erscheinungsjahr, Name der Institution und Nummer des Berichts.

Referenz auf einen Beitrag in einem Buch: diese ist charakterisiert durch Angabe von Autor, Titel des Beitrags, Erscheinungsjahr, Buchtitel und Herausgeber.

Referenz auf einen Beitrag in einer Zeitschrift: diese ist charakterisiert durch Angabe von Autor, Titel des Beitrags, Erscheinungsjahr, Zeitschriftentitel und Ausgabennummer.

6.1 Entwerfen Sie eine geeignete Klassenhierarchie, um die oben aufgelisteten Arten an Literaturreferenzen zu realisieren.

Achten Sie dabei darauf, daß gemeinsame Merkmale in abstrakten Klassen zusammen-gezogen werden. Notieren Sie Ihren Entwurf grafisch und verwenden Sie dazu die folgende Notation (geben Sie lediglich pro Klasse den Namen der Klassen und die Bezeichner der Exemplarvariablen an).



6.2 Geben Sie die Deklaration der Objekttypen (keine geschützten Objekttypen) für die Klassen `ReferenzAufBuch` und `ReferenzAufBeitragInZeitschrift` an. Dabei soll berücksichtigt werden, daß alle Objekte der Klassenhierarchie die Nachrichten `initialisiere` (initialisiert alle Merkmale einer Literaturreferenz mit Standard-Werten) und `alsText` (liefert einen TEXT zurück, der die textuelle Repräsentation aller Merkmale einer Literaturreferenz enthält) kennen müssen. Geben Sie für die beiden Klassen (Objekttypen) auch die notwendigen Zugriffsmethoden für deren spezielle Exemplarvariablen an.

Hinweis: Verwenden Sie der Einfachheit halber ausschließlich TEXT als Typ für die Exemplarvariablen der Klassen.

```
TYPE ReferenzAufBuch =
       LiteraturRef OBJECT
         verlag: TEXT;

       METHODS
         setzeVerlag(name: TEXT) := SetzeVerlag;
         gibVerlag(): TEXT       := GibVerlag;

       OVERRIDES
         initialisiere() := Inititialisiere;
         alsText(): TEXT := AlsText;

       END;


     ReferenzAufBeitragInZeitschrift =
       ReferenzAufBeitrag OBJECT
         nummerAusgabe     : TEXT;

       METHODS
         setzeNummerAusgabe(nr: TEXT) := SetzeNummerAusgabe;
         gibNummerAusgabe(): TEXT     := GibNummerAusgabe;

       OVERRIDES
         initialisiere() := Initialisiere;
         alsText(): TEXT := AlsText;
       END;
```

6.3 Implementieren Sie die zur Nachricht `alsText` gehörende Prozedur der Klasse `ReferenzAufBeitragInZeitschrift.`

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

Aufgabe 7: Lisp

In dieser Aufgabe seien Mengen von Zahlen identisch mit aufsteigend geordneten Listen von Zahlen.
Z. B. entspricht hierbei die Menge {8, 3, 6, 2} der Liste (2, 3, 6, 8) und die Menge {9, 7, 1, 4, 3} der Liste (1, 3, 4, 7, 9). Die Vereinigung der beiden Mengen entspricht der Liste (1, 2, 3, 4, 6, 7, 8, 9).

7.1 Implementieren Sie in LISP eine Funktion `vereinigungs-menge`, die die Vereinigung zweier solcher Mengen berechnet.

```
(defun vereinigungs-menge (m1 m2)

;; wenn eine der beiden Mengen leer ist, dann ist die Vereinigung
;; gleich der anderen Menge

  (cond ((null m1) m2)
        ((null m2) m1)
;; lokale Variablen für die ersten Elemente der Listen

        (T (let ((x1 (car m1)) (x2 (car m2)))

;; Die Vereinigung hängt von dem Verhältnis der ersten Elemente
;; zueinander ab
;; Bei Gleichheit darf das Element nur einmal in der die Vereinigung
;; repräsentierenden Liste auftreten. Deren erstes Element ist x1
;; bzw. x2 und der Rest ergibt aus der Vereinigung der beiden Reste

            (cond ((= x1 x2) (cons x1 (vereinigungs-menge (cdr m1)
                                                          (cdr m2))))

;; Ist x1 kleiner als x2, dann sind alle Elemente der zweiten Liste
;; größer als die der ersten. Somit auch die der die Vereingung
;; repräsentierenden Elemente. Somit ist x1 das erste Element der
;; Ergebnisliste.

                  ((< x1 x2) (cons x1 (vereinigungs-menge (cdr m1)
                                                          m2)))

;; Es bleibt noch der Fall (< x2 x1) übrig. Dieser ist symmetrisch
;; zum vorhergehenden Fall.

                  (T (cons x2 (vereinigungs-menge m1 (cdr m2)))))))) )
```

7.2 Schätzen Sie den zeitlichen Aufwand Ihrer Funktion für die Berechnung der Vereinigungsmenge ab.
Ist der Aufwand Ihrer Funktion nicht in *O(n)*, so überlegen Sie, wie sich eine Funktion schreiben läßt, die die Vereinigung in der Zeit *O(n)* berechnet.

Mit jedem Rekursionsschritt wird das Problem der Mengenvereingung auf die Vereinigung echt kleinerer Mengen reduziert, bis an mindestens einer Stelle die leere Menge erreicht ist. Genauer wird aus mindestens einer Menge ein Element eliminiert. Somit ist die Anzahl der (Rekursions-)Schritte höchstens gleich der Summe der Anzahl der Elemente der Mengen, also |m1|+|m2| (genauere Abschätzung: min{|m1|, |m2|}+1).
Daraus folgt, daß die Vereinigung in der Zeit O(n) berechnet wird.
Überlegungen zur Optimierung sind also nicht notwendig.