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!

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.

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.

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!

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

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`!

Vorname	Name	Matr.-Nr

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.

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)

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!

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.

Vorname	Name	Matr.-Nr

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

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

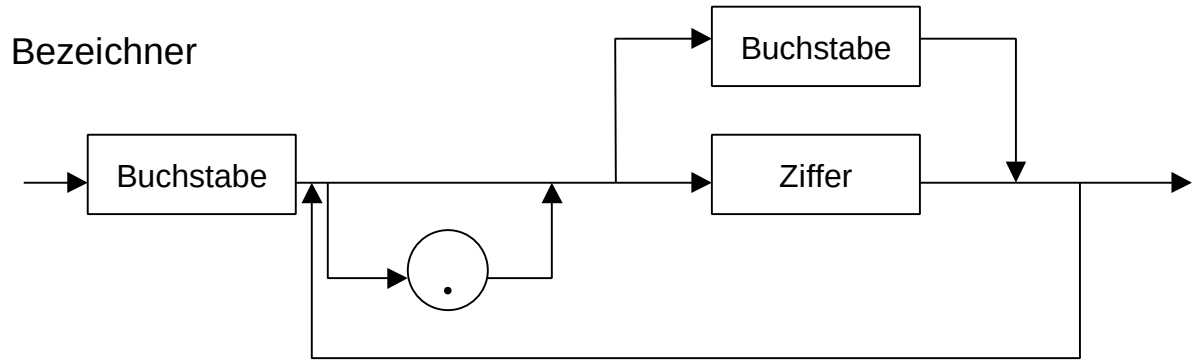
Aufgabe 1 : Syntaxdiagramm und EBNF

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	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	We = X=+1
Wb = OOOOOOX	Wf = X
Wc = O+OXO+O	Wg = +O+O+
Wd = XOOOOO+O+OOXO+OX	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.

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!

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.

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.

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.

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			
2			
3			
4			
5			
6			
7			

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.

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.

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.

Vorname	Name	Fach	Sem.	Matr.-Nr

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

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

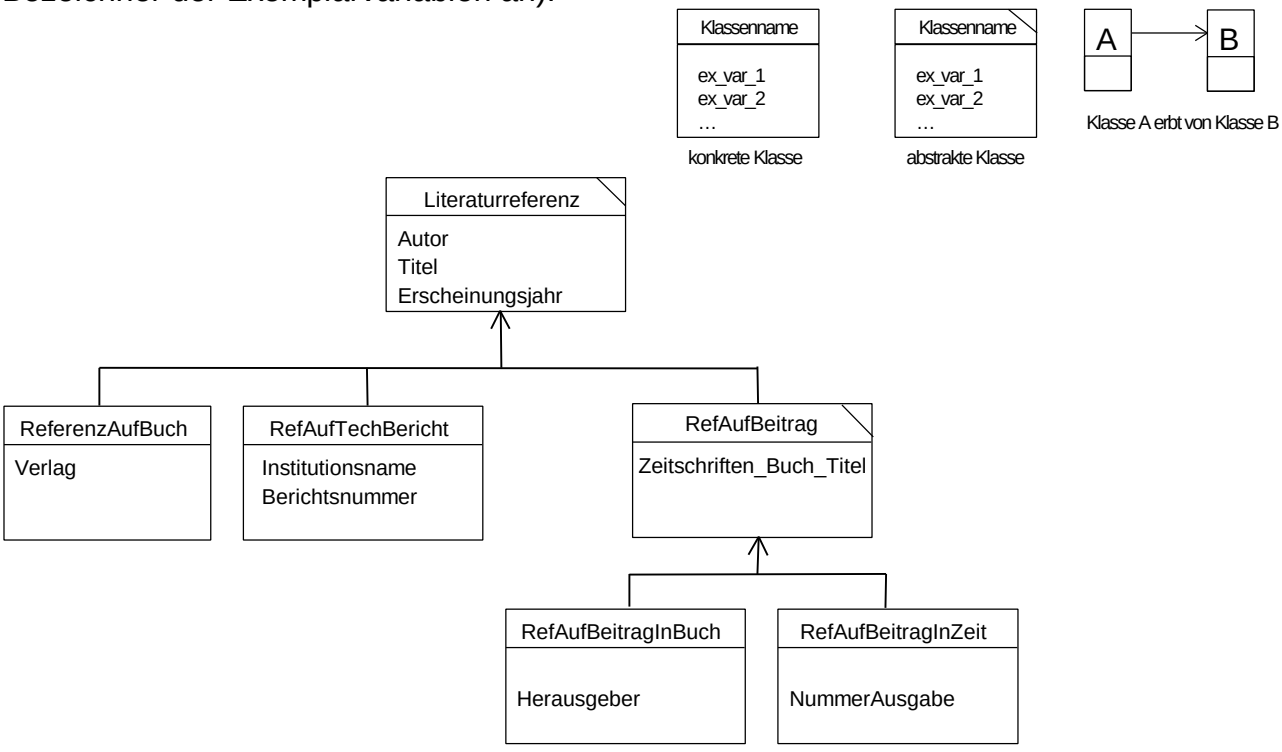
Referenz auf einen Technischen Bericht: diese ist charakterisiert durch Angabe von Autor, Titel, Erscheinungsjahr, Name der Institution und Nummer des Berichts.

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.`

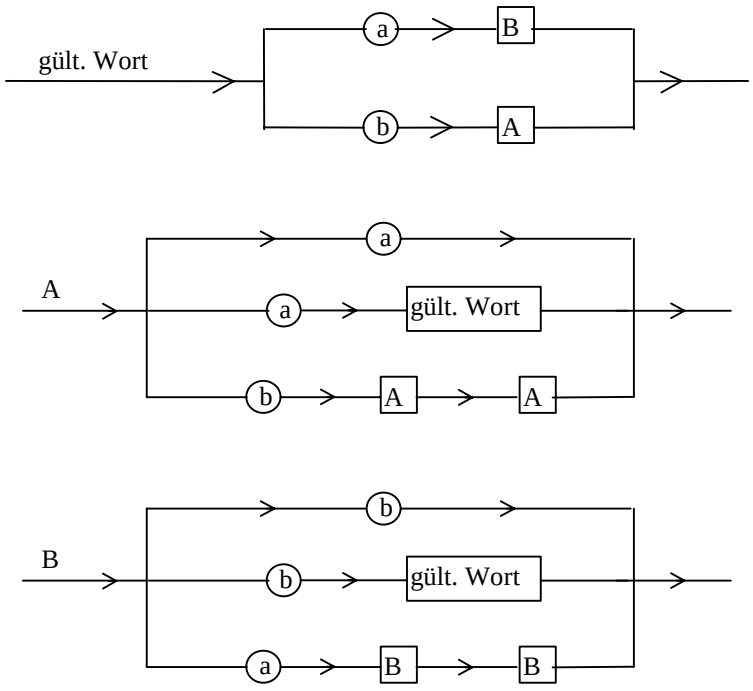
```
PROCEDURE AlsText(self: ReferenzAufBeitragInZeitschrift): TEXT=
BEGIN
  RETURN ReferenzAufBeitrag.alsText() & ", " &
         self.gibNummerAusgabe();
END AlsText;
```

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

Loesung SCHEIN

Aufgabe 1.1 (4 Pkt)

L(G) ist gleich der Sprache, die durch das Syntaxdiagramm für "gült. Wort" beschrieben wird.



Aufgabe 1.2 (4 Pkt)

$L(G) = \{w \in \{a, b\}^* : |w| > 0, |w|_a = |w|_b \}$

Begündung:

Die Bedingung $|w| > 0$ muß stets erfüllt sein, da in jeder von S ausgehenden Produktionsfolge mindestens ein a oder ein b auftritt (da auf S nur die Produktionen 1) und 2) angewandt werden können).

Die Grammatik G besteht aus drei Teilen:
Teil 1: Die linke Seite der Produktionen ist S
Teil 2: Die linke Seite der Produktionen ist A.
Teil 3: Die linke Seite der Produktionen ist B.
Dabei sind die Struktur des zweiten und des dritten Teils gleich, jedoch sind A und B sowie a und b vertauscht.
Der erste Teil besteht aus zwei Regeln, für die das Gleiche gilt wie für den ersten und den zweiten Teil.

Weiterhin kann eine Produktionsfolge nur terminieren, wenn das Nichtterminal A durch a ersetzt und B durch b.
In einer mit A beginnenden Produktionfolge kann nur ein b erzeugt werden, wenn gleichzeitig zwei a's erzeugt werden. Um das Nichtterminal A (von S ausgehend) zu erhalten, muß jedoch zuvor ein b erzeugt werden. Aus A kann direkt (d.h. in einem Produktionsschritt) nur ein a erzeugt werden.
Das Analoge gilt für B (mit A und B bzw. a und b vertauscht).
Somit bedingt die Erzeugung eines a's die Erzeugung genau eines b's und umgekehrt. Für die Reihenfolge der a's und b's gibt es keine Bedingungen.

Aufgabe 2.1 (5 Pkt)

Arbeitsweise:
So lange es möglich ist, werden Schienen der Länge 5 ausgewählt, danach Schienen der Länge 2, schließlich noch Schienen der Länge 1.

```
PROCEDURE Schienen(laenge: CARDINAL) : CARDINAL =
BEGIN
  IF laenge >= 5 THEN RETURN Schienen(laenge-5) + 1;
  ELSIF laenge >= 2 THEN RETURN Schienen(laenge-2) + 1;
  ELSIF  laenge = 1 THEN RETURN 1;
  ELSIF  laenge = 0 THEN RETURN 0;
  END;
END Schienen;
```

Aufgabe 2.2 (3 Pkt)

Arbeitsweise:
Wähle so viele Schienen der Länge 5 wie möglich und danach so viele Schienen der Länge 2 wie möglich. Der Rest wird mit Schienen der Länge 1 aufgefüllt.

```
PROCEDURE SchienenIt(laenge:CARDINAL):CARDINAL =
VAR rest5, rest2 :  CARDINAL;
BEGIN
  rest5 := laenge MOD 5;
  rest2 := laenge MOD 2;
  RETURN  ((laenge DIV 5) + (rest5 DIV 2) + rest2);
END SchienenIt;
```

Aufgabe 3.1 (4 Pkt)

```
;; Prozedur zur Einfuegen eines neuen Mitarbeiters

(defun einfuegen (vorname name lohnform eintritt verdienst)
   (setq PersListe (cons (list (list vorname name)
                               lohnform eintritt verdienst)
                         PersListe)
   )
)
```

Aufgabe 3.2 (4 Pkt)

```
;; Rekursive Funktion, welche die Gesamtsumme der zu zahlenden
```

```
;; Loehne und Gehaelter ermittelt

(defun summe (liste)
   (cond ((null liste) 0)
         (T            (+ (cadddr (car liste))
                          (summe  (cdr liste))
                       )
         )
   )
)
```

Aufgabe 4.1 (3 Pkt)

```
TYPE Container = RECORD
                   idnummer   : CARDINAL;
                   gewicht    : REAL;
                   eigentuemer: TEXT;
                 END;

     Stapelelement = REF RECORD
                           container: Container;

                           (* Verweis auf naechstes Element *)
                           nElement : Stapelelement;
                         END;

     Stapel = REF RECORD
                   stapel : Stapelelement;

                   (* Verweis auf naechsten Stapel *)
                   nStapel: Stapel;
                 END;
```

Aufgabe 4.2 (4 Pkt)

```
PROCEDURE DruckeContainer(c: Container) =
BEGIN
  SIO.PutText("ID-Nr. ");        SIO.PutInt(c.idnummer);
  SIO.PutText(" Gewicht: ");     SIO.PutReal(c.gewicht);
  SIO.PutText(" Eigentuemer: "); SIO.PutText(c.eigentuemer);
END DruckeContainer;


PROCEDURE DruckeContainerWand(ersterStapel: Stapel) =
VAR aktElement: Stapelelement;
    aktStapel : Stapel;
    x, y      : CARDINAL;
BEGIN
  aktStapel := ersterStapel; x := 1;
  WHILE aktStapel # NIL DO

    aktElement := aktStapel^.stapel; y := 1;
    WHILE aktElement # NIL DO
      SIO.PutText("Stapel: "); SIO.PutInt(x);
```

```
      SIO.PutText(" Ebene: "); SIO.PutInt(y);
      SIO.PutText("  ");
      DruckeContainer(aktElement^.container); SIO.Nl();
      aktElement := aktElement^.nElement; INC(y);
    END;

    aktStapel := aktStapel^.nStapel; INC(x);
  END;
END DruckeContainerWand;
```

Aufgabe 5.1 (4 Punkte)

```
INTERFACE Konto;

IMPORT Buchung;
IMPORT Datum;

PROCEDURE Init();
(* Initialisiert das Konto und löscht alle Buchungen *)

PROCEDURE GetInhaber(): TEXT;
(* Liefert den Kontoinhaber *)

PROCEDURE SetInhaber(n: TEXT);
(* Setzt den Inhaber *)

PROCEDURE GetKtoNummer(): TEXT;
(* Liefert die Kontonummer *)

PROCEDURE SetKtoNummer(nr: TEXT);
(* Setzt die Kontonummer *)

PROCEDURE GetBLZ(): TEXT;
(* Liefert die Bankleitzahl *)

PROCEDURE SetBLZ(b: TEXT);
(* Setzt die Bankleitzahl *)

PROCEDURE BerechneSaldo(): REAL;
(* Berechnet den Saldo und gibt diesen als REAL-Wert zurück *)

PROCEDURE BuchungAnhaengen(buchung: Buchung.T);
(* Fügt die übergebene Buchung hinten an die Buchungsliste an
*)

PROCEDURE BuchungLoeschen(buchung: Buchung.T);
(* Löscht die übergebene Buchung aus der Buchungsliste *)

PROCEDURE Kontoauszug(anfang: Datum.T; ende: Datum.T): TEXT;
(* Gibt alle Buchungen, deren Datum innerhalb der übergebenen
Intervallgrenzen liegen, als Text zurück *)

END Konto.
```

Aufgabe 5.2 (2 Punkte)

```
MODULE Konto;

IMPORT Buchung;
IMPORT Datum;

TYPE Buchungsliste = REF RECORD
                       buchung: Buchung.T;
                       next: Buchungsliste;
                     END;

VAR inhaber,ktonummer, blz: TEXT;
    buchungen: Buchungsliste;
```

Aufgabe 5.3 (3 Punkte)

```
PROCEDURE BerechneSaldo(): REAL =
VAR blist: Buchungsliste;
    saldo: REAL;
    b : Buchung.T;
BEGIN
  saldo := 0.0;
  blist := buchungen;
  WHILE (blist # NIL) DO
    b := blist^.buchung;
    saldo := saldo + Buchung.GetBetrag(b);
    blist := blist^.next;
  END;
  RETURN saldo;
END BerechneSaldo;
```

Aufgabe 6.1 (2 Punkte)

```
d_flug(london, berlin, 12, 14).
d_flug(berlin, moskau, 15, 19).
d_flug(london, paris, 6, 8).
d_flug(paris, berlin, 9, 11).
d_flug(paris, madrid, 14, 17).
d_flug(paris, rom, 10, 12).
d_flug(rom, athen, 14, 16).
d_flug(athen, moskau, 18, 24).
```

```
flug(Von, Von).
flug(Von, Nach) :- d_flug(Von, Ueber, _, _),
                   flug(Ueber, Nach).
```

Aufgabe 6.2 (3 Punkte)

```
flugzeit(Von,Nach, Flugzeit) :-
          d_flug(Von,Ueber,Start, Ankunft),
          Dauer is Ankunft-Start,
          weiter_flug(Ueber,Nach,Ankunft,Restflugzeit),
          Flugzeit is Dauer+Restflugzeit.

weiter_flug(Von,Von,_,0).
weiter_flug(Von,Nach,Ankunft, Flugzeit) :-
       d_flug(Von,Ueber,StartWeiterflug,AnkunftWeiterflug),
       Diff is StartWeiterflug-Ankunft,
       Diff>=1,
       Diff=<2,
       Dauer is AnkunftWeiterflug-StartWeiterflug,
       weiter_flug(Ueber,Nach,AnkunftWeiterflug,Restflugzeit),
       Flugzeit is Dauer+Restflugzeit.
```

Vorname	Name	Matr.-Nr	1 schein /L

Scheinklausur "Programmierung"

5.3.2001, Prof. H. Lichter

Hinweise zur Bearbeitung:

- Schreiben Sie auf jedes Blatt **Vorname**, **Name** und **Matrikelnummer**
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- 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	**8**	
Aufgabe 2	**8**	
Aufgabe 3	**8**	
Aufgabe 4	**7**	
Aufgabe 5	**9**	
Aufgabe 6	**5**	
SUMME	**45**	

Vorname	Name	Matr.-Nr	2 schein /L

Aufgabe 1: Grammatik/Sprache

Betrachten Sie die Grammatik G = ({S, A, B}, {a, b}, P, S), wobei P gegeben ist durch:

S → aB	**A → a**	**B → b**
S → bA	**A → aS**	**B → bS**
	A → bAA	**B → aBB**

1.1 (4 Pkt) Geben Sie Syntaxdiagramme an, die die gleiche Sprache wie G beschreiben.

1.2 (4 Pkt) Bestimmen Sie L(G). Geben Sie eine umgangssprachliche, aber genaue Begründung an.

Vorname	Name	Matr.-Nr	3 schein /L

Aufgabe 2: Einfache Funktionen

Eine Eisenbahngesellschaft möchte mehrere neue Schienenstrecken verlegen. Ein Hersteller bietet Schienenstücke der Längen 1m, 2m und 5m an.

2.1 (5 Pkt) Schreiben Sie eine **rekursive** Modula3-Funktion, die zu einer Strecke (gegeben durch eine natürliche Zahl, die die Länge der Strecke in Meter angibt) die minimale Anzahl von Schienenstücken berechnet, die benötigt werden, um die Strecke zu überdecken.

2.2 (3 Pkt) Geben Sie eine **nicht-rekursive** Modula3-Funktion an, die das Gleiche leistet wie die unter 2.1) entwickelte Funktion.

Vorname	Name	Matr.-Nr	4 schein /L

Aufgabe 3: Lisp

Gegeben ist die folgende Listenstruktur, die die Personaldaten eines Unternehmens speichert. Für jeden Mitarbeiter wird der Name, die Gehaltsart, das Eintrittsjahr und das monatliche Gehalt erfasst. Diese Listenstruktur sei in der freien Variablen `PersListe` gespeichert.

```
(((Peter Maier)    Lohn     1965 1200)
 ((Frank Mueller)  Gehalt   1989 1800)
 ((Pia Schmidt)    Lohn     1975 1450)
)
```

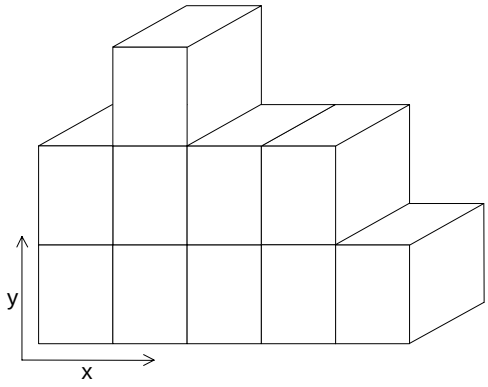
3.1 (4 Pkt) Schreiben Sie eine LISP-Funktion `einfuegen`, die als Parameter den Vornamen, den Namen, die Entlohnungsform (Lohn/Gehalt), das Eintrittsjahr und den momentanen Verdienst eines neuen Mitarbeiters erhält und die in `PersListe` gespeicherte Personalliste um einen entsprechenden Eintrag erweitert.

3.2 (4 Pkt) Es ist Monatsende und die Löhne und Gehälter sind fällig. Schreiben Sie zu diesem Zweck eine rekursive LISP-Funktion `summe`, welche die Gesamtsumme der durch das Unternehmen zu zahlenden Löhne und Gehälter aus der als Parameter übergebenen Personalliste ermittelt.

Vorname	Name	Matr.-Nr	5 schein /L

Aufgabe 4: Zeiger und Listenstrukturen

In einem Containerhafen müssen die zu verladenden Container oft zwischengelagert werden (siehe Abbildung). Dazu können Container zu Stapeln **beliebiger** Höhe (y-Richtung) aufgetürmt werden. Um zusätzlich Platz zu sparen, werden die einzelnen Container-Stapel in einer Reihe (x-Richtung) zu einer Containerwand **beliebiger** Breite aneinander gestellt. Jeder Container ist gekennzeichnet durch eine eindeutige Nummer, sein Gewicht und den Namen des Eigentümers.



4.1 (3 Pkt) Entwerfen Sie geeignete Modula-3 Datenstrukturen, um den Aufbau einer solchen Containerwand im Rechner nachzubilden. Es muss möglich sein, ausgehend vom untersten Container des ersten Stapels (Ursprung des x-y-Achsenkreuzes in der Abbildung) einen beliebigen Container in der Containerwand zu erreichen.

Hinweis:
Entwerfen Sie die Datenstrukturen so, dass die nachfolgende Teilaufgabe damit möglichst einfach implementiert werden kann.

Vorname	Name	Matr.-Nr	6 schein /L

4.2 (4 Pkt) Verwenden Sie im folgenden die von Ihnen in Teil 4.1) entworfenen Datenstrukturen. Schreiben Sie eine Modula-3 Prozedur, welche ausgehend vom untersten Container des ersten Stapels die Daten aller in der Containerwand gelagerten Container mit Positionsangabe (`Stapel: x Ebene: y`) auf dem Bildschirm ausgibt.

Vorname	Name	Matr.-Nr	7 schein /L

Aufgabe 5: Datenabstraktion

In dieser Aufgabe soll eine Kontoverwaltung realisiert werden. Dazu stehen die abstrakten Datentypen `Buchung` und `Datum` zur Verfügung. Der Objektmodul `Konto` ist zu definieren und teilweise zu realisieren. Ein Konto besitzt neben einem Inhaber, einer Kontonummer und einer Bankleitzahl ebenfalls eine Buchungsliste, in der, nach Datum sortiert, alle Buchungen, die dieses Konto betreffen, gespeichert sind. Die Speicherung der Buchungen soll in einer einfach verketteten Liste erfolgen.
Die Schnittstellen der ADTen `Buchung` und `Datum` sind bereits vorgegeben.

```
INTERFACE Buchung;

IMPORT Datum;

TYPE T <: REFANY;

PROCEDURE NeueBuchung(d: Datum.T; b: REAL): T;
(* Erstellt eine neue Buchung mit dem
angegebenen Datum und Betrag (b) *)

PROCEDURE GetDatum(buchung: T): Datum.T;
(* Gibt das Datum der Buchung zurück *)

PROCEDURE GetBetrag(buchung: T): REAL;
(* Gibt den Betrag der Buchung zurück *)

PROCEDURE AlsText(buchung: T): TEXT;
(* Erzeugt eine textuelle Darstellung *)

END Buchung.
```

```
INTERFACE Datum;

TYPE Tag =   [1..31];
     Monat = [1..12];
     Jahr  = [1980..2100];

TYPE T <: REFANY;

PROCEDURE NeuesDatum(t: Tag; m: Monat;
                     j: Jahr): T;
(* Erzeugt ein neues Datum mit den Daten
für Tag, Monat und Jahr*)

PROCEDURE GetTag(datum: T): Tag;
(* Gibt des Tag des Datums zurück *)

PROCEDURE GetMonat(datum: T): Monat;
(* Gibt den Monat des Datums zurück *)

PROCEDURE GetJahr(datum: T): Jahr;
(* Gibt das Jahr des Datums zurück *)

END Datum.
```

5.1 (4 Pkt) Geben Sie die Schnittstelle zum Objektmodul `Konto` an. Deklarieren sie die geforderten Zugriffsprozeduren und fachlichen Funktionen. Insbesondere werden die Funktionen `BerechneSaldo` und `Kontoauszug` benötigt.

`BerechneSaldo` liefert den aktuellen Kontostand auf Basis der vorhandenen Buchungen.
`Kontoauszug` gibt alle Buchungen, die zwischen einem Anfangsdatum und einem Enddatum liegen, als Text zurück.

Vorname	Name	Matr.-Nr	8 schein /L

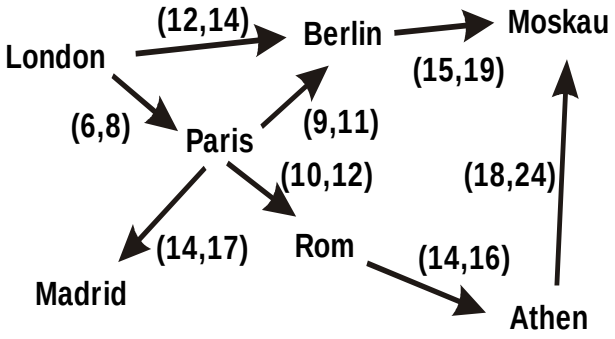
5.2 (2 Pkt) Geben Sie den Deklarationsteil (alle Typ- und Variablendeklarationen) im Implementierungs-Modul (Rumpf) des Objektmoduls `Konto` an.

5.3 (3 Pkt) Implementieren Sie die Funktion `BerechneSaldo`, die den Saldo des Kontos bezüglich der darin gespeicherten Buchungen ermittelt.

Vorname	Name	Matr.-Nr	9 schein /L

Aufgabe 6: Prolog

Die nebenstehende Grafik zeigt die Flugverbindungen zwischen verschiedenen Hauptstädten. Die Verbindungen sind unidirektional, das heißt, eine Flugverbindung existiert nur in Richtung der Pfeilspitze. Zu jedem Flug ist die Ablugzeit und die Ankunftszeit angegeben (alle Städte liegen in der gleichen Zeitzone).
Beispiel: Beim Flug London-Berlin ist die Abflugzeit in London 12.00 Uhr und die Ankunftszeit in Berlin 14.00 Uhr.



6.1 (2 Pkt) Man ist daran interessiert, ob eine Flugverbindung von einem Abflughafen zu einem Zielflughafen existiert. Lösen Sie das Problem mit PROLOG. Beschreiben Sie die vorhandenen Flugverbindungen und definieren Sie ein Prädikat `flug`, welches ermittelt, ob der Zielflughafen vom Startflughafen aus erreichbar ist.

6.2 (3 Pkt) Definieren Sie auf Basis der vorhandenen Flugdaten ein Prädikat `flugzeit`, das ausgehend von einem Startflughafen die reine Flugdauer bis zur Landung am Zielflughafen berechnet. Dabei soll die folgende Bedingung gelten: Es dürfen nur solche Anschlussflüge auf dem Weg zum Zielflughafen genutzt werden, für die gilt, dass zwischen Landung und Weiterflug wenigstens eine und maximal zwei Stunden liegen.

Vorname	Name	Matr.-Nr

Studiengang Technische Redaktion
Zwischenprüfung Grundlagen der Informatik
Teilklausur Programmierung

5. 3.2001, Prof. H. Lichter

Hinweise zur Bearbeitung:

- Schreiben Sie auf jedes Blatt **Vorname**, **Name** und **Matrikelnummer**
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- 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	**6**	
Aufgabe 2	**8**	
Aufgabe 3	**7**	
Aufgabe 4	**7**	
Aufgabe 5	**9**	
Aufgabe 6	**3**	
SUMME	**40**	

Vorname	Name	Matr.-Nr

Aufgabe 1: Grammatik/Sprache

Betrachten Sie die Grammatik G = ({S, A, B}, {a, b}, P, S), wobei P gegeben ist durch:

1) **S → aB**
2) **S → bA**
3) **A → a**
4) **A → aS**
5) **A → bAA**
6) **B → b**
7) **B → bS**
8) **B → aBB**

1.1 (4 Pkt) Geben Sie zu jedem der folgenden Wörter an, ob es in der Sprache L(G) enthalten ist und geben Sie die Reihenfolge der verwendeten Produktionen (siehe Beispiel in der ersten Zeile) an.

Wort	enthalten	nicht enthalten	Produktionsfolge, wenn enthalten
ab	X		1, 6
aabb			
abbaba			
ababa			
abbaa			

1.2 (2 Pkt) Erläutern Sie möglichst präzise die Begriffe „Formale Sprache“ und „kontextfreie Grammatik“.

Vorname	Name	Matr.-Nr	3 tr/L

Aufgabe 2: Einfache Funktionen

Eine Eisenbahngesellschaft möchte mehrere neue Schienenstrecken verlegen. Ein Hersteller bietet Schienenstücke der Längen 1m, 2m und 5m an.

2.1 (5 Pkt) Schreiben Sie eine **rekursive** Modula3-Funktion, die zu einer Strecke (gegeben durch eine natürliche Zahl, die die Länge der Strecke in Meter angibt) die minimale Anzahl von Schienenstücken berechnet, die benötigt werden, um die Strecke zu überdecken.

2.2 (3 Pkt) Geben Sie eine **nicht-rekursive** Modula3-Funktion an, die das Gleiche leistet wie die unter 2.1) entwickelte Funktion.

Vorname	Name	Matr.-Nr	4 tr/L

Aufgabe 3: Lisp

Gegeben ist die folgende Listenstruktur, die die Personaldaten eines Unternehmens speichert. Für jeden Mitarbeiter wird der Name, die Gehaltsart, das Eintrittsjahr und das monatliche Gehalt erfasst. Diese Listenstruktur sei in der freien Variablen `pd` gespeichert.

```
(((Peter Maier)    Lohn     1965 1200)
 ((Frank Mueller)  Gehalt   1989 1800)
 ((Pia Schmidt)    Lohn     1975 1450)
)
```

3.1 (3 Pkt) Geben Sie jeweils das Ergebnis der folgenden LISP-Ausdrücke an:

Ausdruck 1:
```
(car (cdr (car pd)))
```

Ergebnis: _____

Ausdruck 2:
```
(+ (cadddr (car   pd))
   (cadddr (cadr  pd))
   (cadddr (caddr pd)))
```

Ergebnis: _____

Ausdruck 3:
```
(cons (cadar (caddr pd))
      (cons (cadar (cadr pd))
            (cons (cadaar pd) ())
       )
 )
```

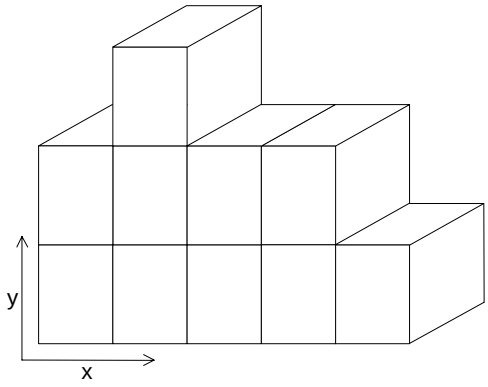
Ergebnis: _____

3.2 (4 Pkt) Schreiben Sie eine LISP-Funktion `einfuegen`, die als Parameter den Vornamen, den Namen, die Entlohnungsform (Lohn/Gehalt), das Eintrittsjahr und den momentanen Verdienst eines neuen Mitarbeiters erhält und die in `pd` gespeicherte Personalliste um einen entsprechenden Eintrag erweitert.

Vorname	Name	Matr.-Nr	5
			tr/L

Aufgabe 4: Zeiger und Listenstrukturen

In einem Containerhafen müssen die zu verladenden Container oft zwischengelagert werden (siehe Abbildung). Dazu können mehrere Container zu Stapeln **beliebiger** Höhe (y-Richtung) aufgetürmt werden. Um zusätzlich Platz zu sparen, werden die einzelnen Container-Stapel in einer Reihe (x-Richtung) zu einer Containerwand **beliebiger** Breite aneinander gestellt. Jeder Container ist gekennzeichnet durch eine eindeutige Nummer, sein Gewicht und den Namen des Eigentümers.



4.1 (3 Pkt) Entwerfen Sie geeignete Modula-3 Datenstrukturen, um den Aufbau einer solchen Containerwand im Rechner nachzubilden. Es muss möglich sein, ausgehend vom untersten Container des ersten Stapels (Ursprung des x-y-Achsenkreuzes in der Abbildung) einen beliebigen Container in der Containerwand zu erreichen.

Hinweis:
Entwerfen Sie die Datenstrukturen so, dass die nachfolgende Teilaufgabe damit möglichst einfach implementiert werden kann.

Vorname	Name	Matr.-Nr

4.2 (4 Pkt) Verwenden Sie im folgenden die von Ihnen in Teil 4.1) entworfenen Datenstrukturen. Schreiben Sie eine Modula-3 Prozedur, welche ausgehend vom untersten Container des ersten Stapels die Daten aller in der Containerwand gelagerten Container mit Positionsangabe (`Stapel: x Ebene: y`) auf dem Bildschirm ausgibt.

Vorname	Name	Matr.-Nr	7
			tr/L

Aufgabe 5: Datenabstraktion

In dieser Aufgabe soll eine Kontoverwaltung realisiert werden. Dazu stehen die abstrakten Datentypen `Buchung` und `Datum` zur Verfügung. Der Objektmodul `Konto` ist zu definieren und teilweise zu realisieren. Ein Konto besitzt neben einem Inhaber, einer Kontonummer und einer Bankleitzahl ebenfalls eine Buchungsliste, in der, nach Datum sortiert, alle Buchungen, die dieses Konto betreffen, gespeichert sind. Die Speicherung der Buchungen soll in einer einfach verketteten Liste erfolgen.
Die Schnittstellen ADTen `Buchung` und `Datum` sind bereits vorgegeben.

```
INTERFACE Buchung;

IMPORT Datum;

TYPE T <: REFANY;

PROCEDURE NeueBuchung(d: Datum.T; b: REAL): T;
(* Erstellt eine neue Buchung mit dem
angegebenen Datum und Betrag (b) *)

PROCEDURE GetDatum(buchung: T): Datum.T;
(* Gibt das Datum der Buchung zurück *)

PROCEDURE GetBetrag(buchung: T): REAL;
(* Gibt den Betrag der Buchung zurück *)

PROCEDURE AlsText(buchung: T): TEXT;
(* Erzeugt eine textuelle Darstellung *)

END Buchung.
```

```
INTERFACE Datum;

TYPE Tag =   [1..31];
     Monat = [1..12];
     Jahr  = [1980..2100];

TYPE T <: REFANY;

PROCEDURE NeuesDatum(t: Tag; m: Monat;
                     j: Jahr): T;
(* Erzeugt ein neues Datum mit den Daten
für Tag, Monat und Jahr*)

PROCEDURE GetTag(datum: T): Tag;
(* Gibt des Tag des Datums zurück *)

PROCEDURE GetMonat(datum: T): Monat;
(* Gibt den Monat des Datums zurück *)

PROCEDURE GetJahr(datum: T): Jahr;
(* Gibt das Jahr des Datums zurück *)

END Datum.
```

5.1 (4 Pkt) Geben Sie die Schnittstelle zum Objektmodul `Konto` an. Deklarieren sie die geforderten Zugriffsprozeduren und fachlichen Funktionen. Insbesondere werden die Funktionen `BerechneSaldo` und `Kontoauszug` benötigt.

`BerechneSaldo` liefert den aktuellen Kontostand auf Basis der vorhandenen Buchungen.
`Kontoauszug` gibt alle Buchungen, die zwischen einem Anfangsdatum und einem Enddatum liegen, als Text zurück.

Vorname	Name	Matr.-Nr	8
			tr/L

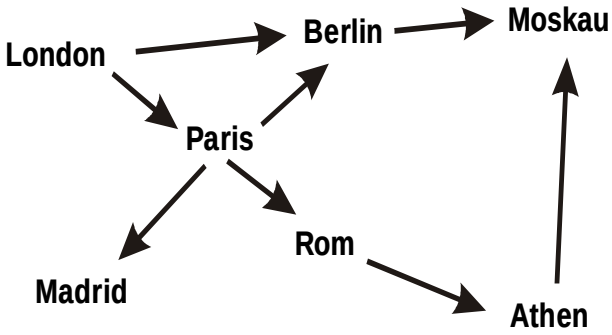
5.2 (2 Pkt) Geben Sie den Deklarationsteil (alle Typ- und Variablendeklarationen) im Implementierungs-Modul (Rumpf) des Objektmoduls `Konto` an.

5.3 (3 Pkt) Implementieren Sie die Funktion `BerechneSaldo`, die den Saldo des Kontos bezüglich der darin gespeicherten Buchungen ermittelt.

Vorname	Name	Matr.-Nr	9 tr/L

Aufgabe 6: Prolog

Die nebenstehende Grafik zeigt die Flugverbindungen zwischen verschiedenen Hauptstädten. Die Verbindungen sind unidirektional, das heißt, eine Flugverbindung existiert nur in Richtung der Pfeilspitze.



6.1 (3 Pkt) Man ist daran interessiert, ob eine Flugverbindung von einem Abflughafen zu einem Zielflughafen existiert. Lösen Sie das Problem mit PROLOG. Beschreiben Sie die vorhandenen Flugverbindungen und definieren Sie ein Prädikat `flug`, welches ermittelt, ob der Zielflughafen vom Startflughafen aus erreichbar ist.

Programmierung	WS 2000 / 2001	Prof. H. Lichter

Loesung TR

Aufgabe 1.1 (4 Pkt)

Wort	enthalten	nicht enthalten	Produktionsfolge
ab	X		1, 6
aabb	X		1, 8, 6 (angewandt auf das erste B), 6
abbaba	X		1, 7, 2, 4, 2, 3
ababa		X	
abbaa		X	

Aufgabe 1.2 (2 Pkt)

Alphabet:
Ein Alphabet ist eine nichtleere endliche Menge von unterscheidbaren Zeichen.

Formale Sprache:
Sei A ein Alphabet. Eine formale Sprache (über A) ist eine beliebige Teilmenge von A^*.

Aufgabe 2.1 (5 Pkt)

Arbeitsweise:
So lange es möglich ist, werden Schienen der Länge 5 ausgewählt, danach Schienen der Länge 2, schließlich noch Schienen der Länge 1.

```
PROCEDURE Schienen(laenge: CARDINAL) : CARDINAL =
BEGIN
  IF laenge >= 5 THEN RETURN Schienen(laenge-5) + 1;
  ELSIF laenge >= 2 THEN RETURN Schienen(laenge-2) + 1;
  ELSIF  laenge = 1 THEN RETURN 1;
  ELSIF  laenge = 0 THEN RETURN 0;
  END;
END Schienen;
```

Aufgabe 2.2 (3 Pkt)

Arbeitsweise:
Wähle so viele Schienen der Länge 5 wie möglich und danach so viele Schienen der Länge 2 wie möglich. Der Rest wird mit Schienen der Länge 1 aufgefüllt.

```
PROCEDURE SchienenIt(laenge:CARDINAL):CARDINAL =
VAR rest5, rest2 :  CARDINAL;
BEGIN
  rest5 := laenge MOD 5;
  rest2 := laenge MOD 2;
  RETURN  ((laenge DIV 5) + (rest5 DIV 2) + rest2);
END SchienenIt;
```

Aufgabe 3.1 (3 Pkt)

```
Ausdruck 1: (car (cdr (car pd)))
Ergebnis: LOHN


Ausdruck 2: (+ (cadddr (car   pd))
               (cadddr (cadr  pd))
               (cadddr (caddr pd)))
Ergebnis: 4450 = Summe der Löhne/Gehälter


Ausdruck 3: (cons (cadar (caddr pd))
                  (cons (cadar (cadr pd))
                        (cons (cadaar pd) ())
                  )
            )
Ergebnis: (SCHMIDT MUELLER MAIER) = Liste der Nachnamen
```

Aufgabe 3.2 (4 Pkt)

```
;; Prozedur zur Einfuegen eines neuen Mitarbeiters

(defun einfuegen (vorname name lohnform eintritt verdienst)
   (setq PersListe (cons (list (list vorname name)
                               lohnform eintritt verdienst)
                         PersListe)
   )
)
```

Aufgabe 4.1 (3 Pkt)

```
TYPE Container = RECORD
                   idnummer   : CARDINAL;
                   gewicht    : REAL;
                   eigentuemer: TEXT;
                 END;

     Stapelelement = REF RECORD
                           container: Container;

                           (* Verweis auf naechstes Element *)
                           nElement : Stapelelement;
                         END;

     Stapel = REF RECORD
                    stapel : Stapelelement;

                    (* Verweis auf naechsten Stapel *)
                    nStapel: Stapel;
                  END;
```

Aufgabe 4.2 (4 Pkt)

```
PROCEDURE DruckeContainer(c: Container) =
BEGIN
  SIO.PutText("ID-Nr. ");        SIO.PutInt(c.idnummer);
  SIO.PutText(" Gewicht: ");     SIO.PutReal(c.gewicht);
  SIO.PutText(" Eigentuemer: "); SIO.PutText(c.eigentuemer);
END DruckeContainer;


PROCEDURE DruckeContainerWand(ersterStapel: Stapel) =
VAR aktElement: Stapelelement;
    aktStapel : Stapel;
    x, y      : CARDINAL;
BEGIN
  aktStapel := ersterStapel; x := 1;
  WHILE aktStapel # NIL DO

    aktElement := aktStapel^.stapel; y := 1;
    WHILE aktElement # NIL DO
      SIO.PutText("Stapel: "); SIO.PutInt(x);
      SIO.PutText(" Ebene: "); SIO.PutInt(y);
      SIO.PutText("  ");
      DruckeContainer(aktElement^.container); SIO.Nl();
      aktElement := aktElement^.nElement; INC(y);
    END;

    aktStapel := aktStapel^.nStapel; INC(x);
  END;
END DruckeContainerWand;
```

Aufgabe 5.1 (4 Punkte)

```
INTERFACE Konto;

IMPORT Buchung;
IMPORT Datum;

PROCEDURE Init();
(* Initialisiert das Konto und löscht alle Buchungen *)

PROCEDURE GetInhaber(): TEXT;
(* Liefert den Kontoinhaber *)

PROCEDURE SetInhaber(n: TEXT);
(* Setzt den Inhaber *)

PROCEDURE GetKtoNummer(): TEXT;
(* Liefert die Kontonummer *)

PROCEDURE SetKtoNummer(nr: TEXT);
(* Setzt die Kontonummer *)

PROCEDURE GetBLZ(): TEXT;
(* Liefert die Bankleitzahl *)

PROCEDURE SetBLZ(b: TEXT);
(* Setzt die Bankleitzahl *)

PROCEDURE BerechneSaldo(): REAL;
(* Berechnet den Saldo und gibt diesen als REAL-Wert zurück *)

PROCEDURE BuchungAnhaengen(buchung: Buchung.T);
(* Fügt die übergebene Buchung hinten an die Buchungsliste an
*)

PROCEDURE BuchungLoeschen(buchung: Buchung.T);
(* Löscht die übergebene Buchung aus der Buchungsliste *)

PROCEDURE Kontoauszug(anfang: Datum.T; ende: Datum.T): TEXT;
(* Gibt alle Buchungen, deren Datum innerhalb der übergebenen
Intervallgrenzen liegen, als Text zurück *)

END Konto.
```

Aufgabe 5.2 (2 Punkte)

```
MODULE Konto;

IMPORT Buchung;
IMPORT Datum;

TYPE Buchungsliste = REF RECORD
                       buchung: Buchung.T;
                       next: Buchungsliste;
                     END;
```

```
VAR inhaber,ktonummer, blz: TEXT;
    buchungen: Buchungsliste;
```

Aufgabe 5.3 (3 Punkte)

```
PROCEDURE BerechneSaldo(): REAL =
VAR blist: Buchungsliste;
    saldo: REAL;
    b : Buchung.T;
BEGIN
  saldo := 0.0;
  blist := buchungen;
  WHILE (blist # NIL) DO
    b := blist^.buchung;
    saldo := saldo + Buchung.GetBetrag(b);
    blist := blist^.next;
  END;
  RETURN saldo;
END BerechneSaldo;
```

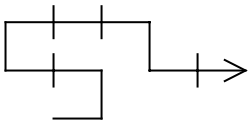
Aufgabe 6.1 (3 Punkte)

```
d_flug(london, berlin, 12, 14).
d_flug(berlin, moskau, 15, 19).
d_flug(london, paris, 6, 8).
d_flug(paris, berlin, 9, 11).
d_flug(paris, madrid, 14, 17).
d_flug(paris, rom, 10, 12).
d_flug(rom, athen, 14, 16).
d_flug(athen, moskau, 18, 24).

flug(Von, Von).
flug(Von, Nach) :- d_flug(Von, Ueber, _, _),
                   flug(Ueber, Nach).
```

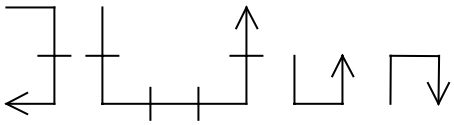
Vorname	Name	Matr.-Nr	1

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.

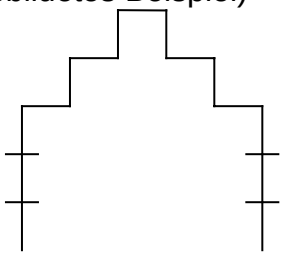
Vorname	Name	Matr.-Nr	2

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.

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

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)



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.

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.

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.

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!

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.

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.

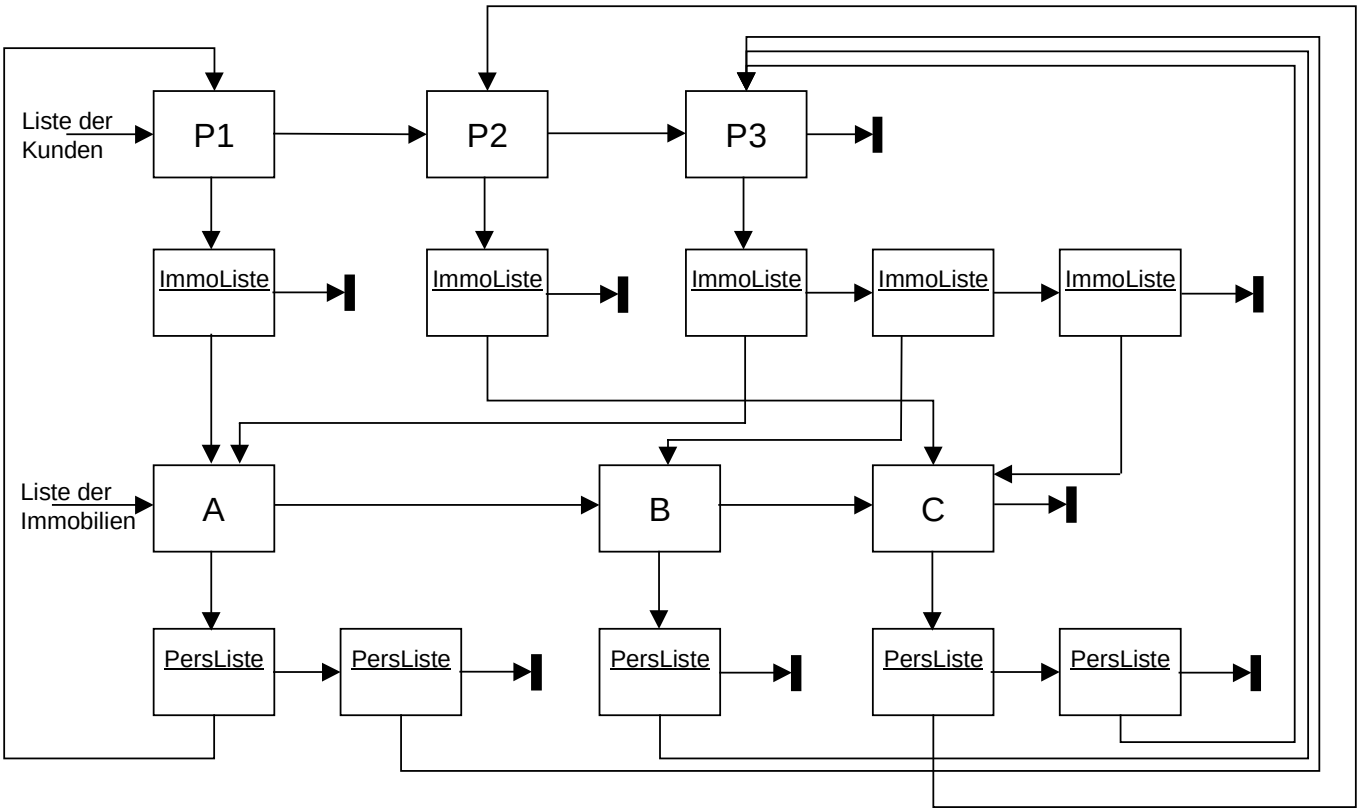
Immobilien- und Kundeninformationen sollen so miteinander verknüpft sein, daß Änderungen an bestehenden Immobilien (z.B. ihr Wert) und Änderungen an Kundeninformationen (z.B. neue Adresse) nur an einer Stelle vorgenommen werden müssen. Wenn also beispielsweise die Kunden A und B die Immobilie C besitzen 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!

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>

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:

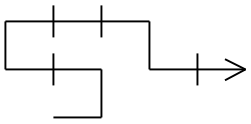
$z = (x_1,y_1, x_2,y_2,\ldots,x_m,y_m,y_{m+1}, \ldots, y_n)$	falls n > m
$z = (x_1,y_1, x_2,y_2,\ldots,x_n,y_n,y_{n+1}, \ldots, y_m)$	falls m > n
$z = (x_1,y_1, x_2,y_2,\ldots,x_m,y_n)$	falls n = m

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)

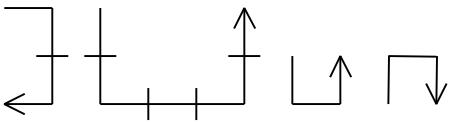
Vorname	Name	Matr.-Nr	1

Aufgabe 1: Syntax

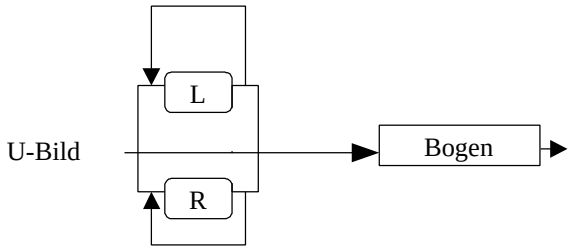
Ein Zeichenkopf werde durch die Anweisungen Links (L), Rechts(R) und Schritt (S) gesteuert. Schritt bewirkt einen Schritt in der bisherigen Richtung, Links und Rechts bewirken eine Drehung um 90 Grad. Alle Schritte sind gleich lang, die Länge eines Schrittes sei a. Zu Beginn ist die Richtung horizontal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

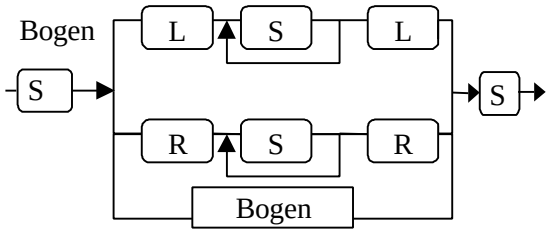


Mit diesen Mitteln kann man auch U-förmige Gebilde erzeugen (siehe nachfolgende Figuren).



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.

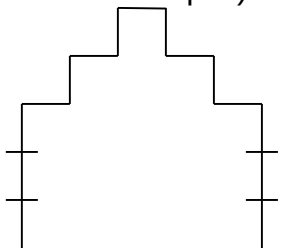
a) Alle Rechtecke mit der Höhe 3*a, Breite n*a, n > 0.

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!

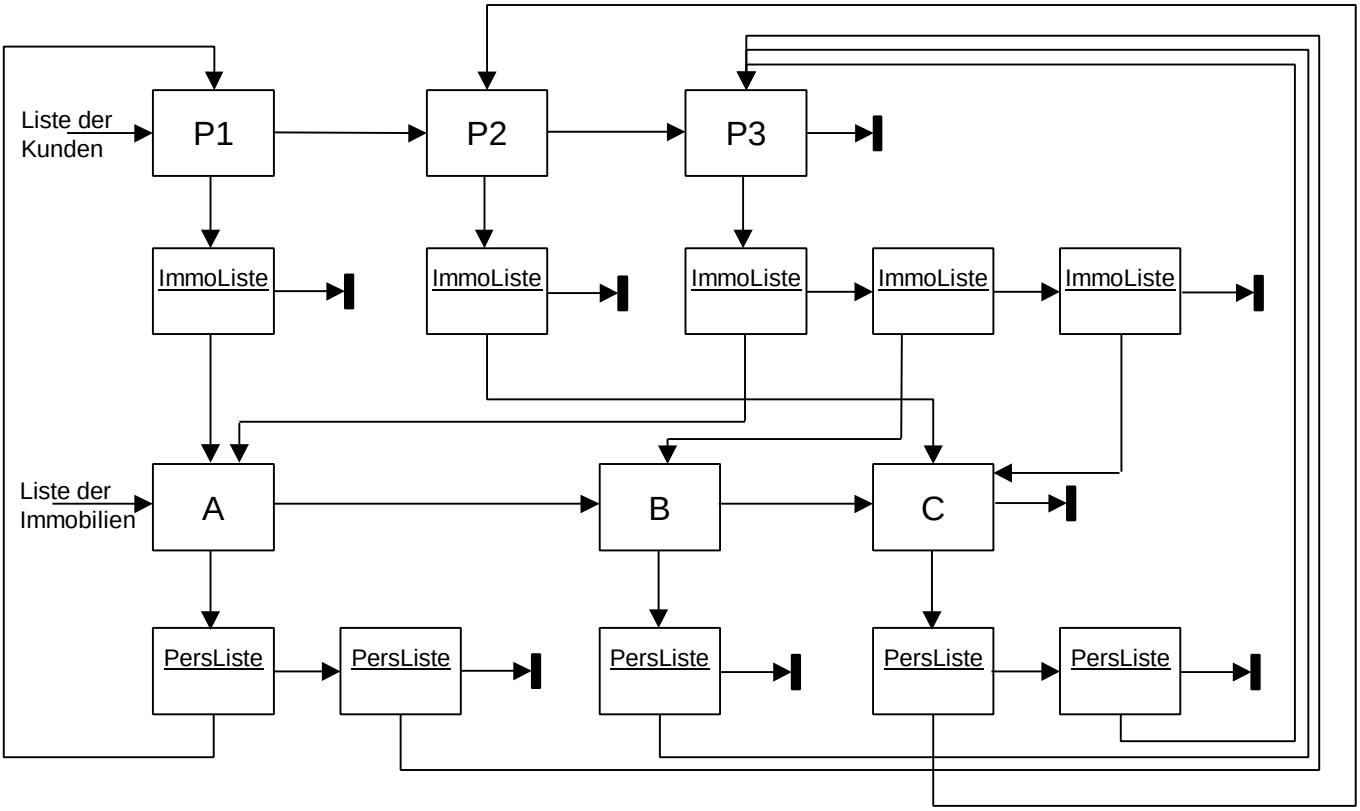
```
TYPE

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;
```

Vorname	Name	Matr.-Nr	1

Aufgabe 1: Syntax

Gegeben sei die folgende Syntaxdefinition für spezielle Zeichenfolgen:

Eingabe = Folge "=".
Folge = Zi | Folge Bu Zi .
Bu = "A" | "B" | "C".
Zi = "1" | "2" | "3".

1.1 (3 Pkt) Welche Zeichenfolgen entsprechen dieser Syntaxdefinition? Bitte ankreuzen!

	entspricht der Syntax	entspricht nicht der Syntax
A B C =		
A 1 3 =		
2 A 1 =		
1 2 3 =		
3 C 2 B 1 =		
1 C 1 C 1 C =		

1.2 (1 Pkt) Geben Sie für die Zeichenfolge 2 B 1 A 3 = den Syntaxbaum an.

1.3 (2 Pkt) Beschreiben Sie die oben angegebene Syntaxdefinition mit einem einzigen Syntaxdiagramm.

Vorname	Name	Matr.-Nr	2

Aufgabe 2: Rekursive Prozeduren

2.1 (5 Pkt) Schreiben Sie eine rekursive Modula-3 Prozedur `Zeile`, die mit Hilfe einer weiteren rekursiven Prozedur `DruckeSterne` für einen Eingabewert n auf 2n-1 Zeilen das folgende zu- und abnehmende Sternmuster ausgibt:
(Bspl: n = 4)

```
*
**
***
****
***
**
*
```

Vorname	Name	Matr.-Nr	3

Aufgabe 3: Zahlen

Eine Eingabezeile, bestehend aus genau 256 Zeichen, soll zeichenweise eingelesen werden. Die eingelesenen Zeichen sind folgendermaßen zu interpretieren. Alle Ziffernfolgen, die keine Leerzeichen enthalten, sind als ganze Zahlen zu interpretieren. Ihr Wert wird nach dem Hornerschema ermittelt.

(z.B.): 1342 = ((1*10+3)*10+4)*10+2)

Die so gefundenen INTEGER-Zahlen sind in einem Feld abzuspeichern.

3.1 (1 Pkt) Geben Sie die Modula-3 Typdeklaration für das Feld an. Dimensionieren Sie das Feld so, daß es die maximal mögliche Anzahl von ganzen Zahlen, die in einer Eingabezeile (=256 Zeichen) geschrieben werden können, aufnehmen kann (und nicht mehr). Begründen Sie Ihre Entscheidung.

3.2 (5 Pkt) Implementieren Sie eine Modula-3 Prozedur, die eine Eingabezeile zeichenweise liest, gemäß obiger Beschreibung interpretiert und die Zahlen in dem deklarierten Feld abspeichert.

Vorname	Name	Matr.-Nr	4

Aufgabe 4: Vier-Gewinnt

Wir möchten ein Vier-Gewinnt-Spiel programmieren. Das Spielbrett steht aufrecht und besteht aus sieben Schächten, die jeweils sechs Steine aufnehmen können. Zwei Spieler spielen gegeneinander, der erste spielt mit schwarzen, der zweite mit roten Steinen. Jeder Spieler versucht, vier eigene Steine lückenlos in einer Zeile, einer Spalte oder in einer Diagonalen zu plazieren, bevor dies dem Gegner gelingt. Dazu werfen die Spieler jeweils abwechselnd einen Stein ihrer Farbe in einen der sieben Schächte. Wird ein Stein in einen Schacht eingeworfen, fällt er bis auf die tiefste noch freie Position dieses Schachtes.

4.1 (2 Pkt) Definieren Sie in Modula-3 eine geeignete Datenstruktur für das Spiel Vier-Gewinnt .
Hinweis: Die Prozeduren der Teilaufgaben 2) und 3) müssen auf der von Ihnen gewählten Datenstruktur operieren.

4.2 (1 Pkt) Deklarieren Sie eine Spielbrett-Variable und initialisieren Sie diese korrekt, damit das Spiel beginnen kann.

Vorname	Name	Matr.-Nr	5

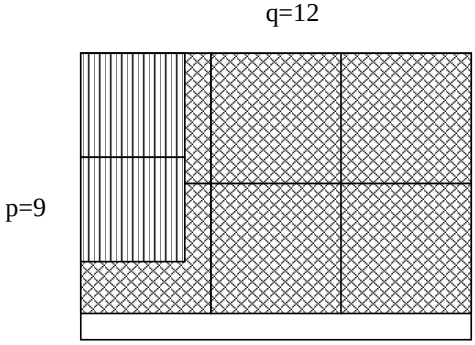
4.3 (4 Pkt) Schreiben Sie eine Modula-3 Prozedur `SteinEinwerfen`, die als Parameter die Angabe des Schachtes und die Steinfarbe erhält. Kann der Stein eingeworfen werden, dann muß der Inhalt des Spielbretts entsprechend verändert sein.

4.4 (6 Pkt) Schreiben Sie eine Modula-3 Funktion `HatGewonnen`, die ermittelt, ob ein Spieler durch Setzen des letzten Steins das Spiel gewonnen hat. Die Funktion erhält als Parameter die Schacht- und Positionsangabe des zuletzt eingeworfenen Steins.

Vorname	Name	Matr.-Nr	6

Aufgabe 5: 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.

5.1 (6 Pkt) 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.

Vorname	Name	Matr.-Nr	7

Aufgabe 6: Punkte und Rechtecke

Wir betrachten in dieser Aufgabe Punkte und achsenparallele Rechtecke in der Ebene. Ein Punkt ist definiert durch ein Koordinatenpaar (x,y). Ein achsenparalleles Rechteck ist definiert durch die linke, untere Ecke und die rechte, obere Ecke. Eine Ecke ist durch ein Koordinatenpaar (x,y) gegeben.

6.1 (6 Pkt) Geben Sie die Interface-Module für die abstrakten Datentypen Punkt und Rechteck an. Nehmen Sie alle die Operationen in die Schnittstellen der abstrakten Datentypen auf, die Sie in den nachfolgenden Teilaufgaben benötigen.

Vorname	Name	Matr.-Nr	8

6.2 (2 Pkt) Geben Sie den Deklarationsteil des Implementierungs-Moduls des ADTs `Rechteck` an (d.h. alles bis zur ersten implementierten Operation).

Gegeben seien eine Liste R von n achsenparallelen Rechtecken und eine Liste P von m Punkten.

6.3 (1 Pkt) Definieren Sie die Typen `ListeVonPunkten` und `ListeVonRechtecken`.

Vorname	Name	Matr.-Nr	9

6.4 (5 Pkt) Schreiben Sie eine Modula-3 Funktion, die ermittelt, in welchem Rechteck der Liste R die meisten Punkte der Liste P liegen. (Liegen in mehreren Rechtecken gleich viele Punkte, so soll die Funktion eines davon zurückgeben)

Vorname	Name	Matr.-Nr	1

Aufgabe 1: Syntax

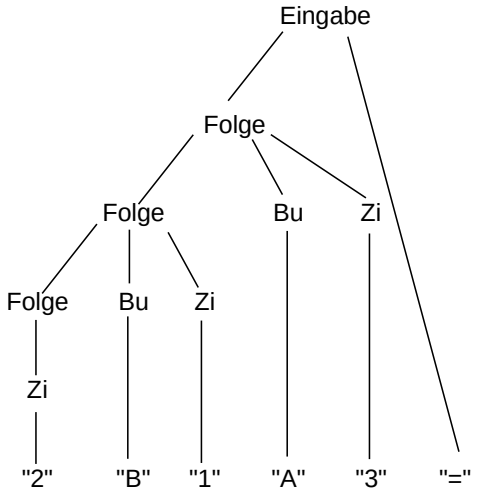
Gegeben sei die folgende Syntaxdefinition für spezielle Zeichenfolgen:

Eingabe = Folge "=".
Folge = Zi | Folge Bu Zi .
Bu = "A" | "B" | "C".
Zi = "1" | "2" | "3".

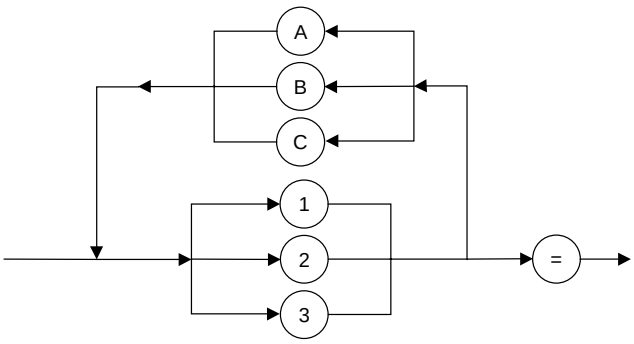
1.1 (3 Pkt) Welche Zeichenfolgen entsprechen dieser Syntaxdefinition? Bitte ankreuzen!

	entspricht der Syntax	entspricht nicht der Syntax
A B C =		X
A 1 3 =		X
2 A 1 =	X	
1 2 3 =		X
3 C 2 B 1 =	X	
1 C 1 C 1 C =		X

1.2 (1 Pkt) Geben Sie für die Zeichenfolge 2 B 1 A 3 = den Syntaxbaum an.



1.3 (2 Pkt) Beschreiben Sie die oben angegebene Syntaxdefinition mit einem einzigen Syntaxdiagramm.



Vorname	Name	Matr.-Nr	2

Aufgabe 2: Rekursive Prozeduren

2.1 (5 Pkt) Schreiben Sie eine rekursive Modula-3 Prozedur `Zeile`, die mit Hilfe einer weiteren rekursiven Prozedur `DruckeSterne` für einen Eingabewert n auf 2n-1 Zeilen das folgende zu- und abnehmende Sternmuster ausgibt:
(Bspl: n = 4)

```
*
**
***
****
***
**
*
```

```
PROCEDURE Zeile(n, r: CARDINAL)=
(* Start mit Zeile(n, 1) definiert fuer n > 0 *)

BEGIN
  (* Maximale Anzahl Rekursionsschritte ist n *)
  IF (r < n) THEN
    DruckeSterne(r);
    Zeile(n, r + 1); (* Rekursion mit r + 1 *)
  END;
  DruckeSterne(r);
END Zeile;


PROCEDURE DruckeSterne(n: CARDINAL)=
BEGIN
  IF (n > 0) THEN
    SIO.PutChar('*');
    DruckeSterne(n - 1); (* Rekursion für n - 1 *)
  ELSE
    SIO.Nl();
  END;
END DruckeSterne;
```

Vorname	Name	Matr.-Nr	3

Aufgabe 3: Zahlen

Eine Eingabezeile, bestehend aus genau 256 Zeichen, soll zeichenweise eingelesen werden. Die eingelesenen Zeichen sind folgendermaßen zu interpretieren. Alle Ziffernfolgen, die keine Leerzeichen enthalten, sind als ganze Zahlen zu interpretieren. Ihr Wert wird nach dem Hornerschema ermittelt.

(z.B.): 1342 = ((1*10+3)*10+4)*10+2)

Die so gefundenen INTEGER-Zahlen sind in einem Feld abzuspeichern.

3.1 (1 Pkt) Geben Sie die Modula-3 Typdeklaration für das Feld an. Dimensionieren Sie das Feld so, daß es die maximal mögliche Anzahl von ganzen Zahlen, die in einer Eingabezeile (=256 Zeichen) geschrieben werden können, aufnehmen kann (und nicht mehr). Begründen Sie Ihre Entscheidung.

```
TYPE IntFeld = ARRAY [1..128] OF INTEGER;
```

Begründung: Zwei INTEGER-Zahlen müssen durch ein Leerzeichen getrennt sein, so daß mit 256 Zeichen maximal 128 einstellige INTEGER-Zahlen darstellbar sind.

3.2 (5 Pkt) Implementieren Sie eine Modula-3 Prozedur, die eine Eingabezeile zeichenweise liest, gemäß obiger Beschreibung interpretiert und die Zahlen in dem deklarierten Feld abspeichert.

```
PROCEDURE Lesen(): IntFeld=
VAR erg        := IntFeld {0,..};
    zaehler    : INTEGER;
    leerzeichen: BOOLEAN; (* Letztes Zeichen? *)
    c          : CHAR;

BEGIN
  zaehler := 0; leerzeichen := TRUE;

  (* Liest genau 256 Zeichen aus Eingabe *)
  FOR i := 1 TO 256 DO
    c := SIO.GetChar();
    IF (c = ' ') THEN
      leerzeichen := TRUE;
    ELSIF (c # ' ') THEN
      IF leerzeichen THEN
        (* Erhoehe Zaehler um 1, da neue Zahl *)
        INC(zaehler);
        leerzeichen := FALSE;
      END;
      erg[zaehler] := erg[zaehler] * 10 + (ORD(c) - ORD('0'));
    END;
  END;

  RETURN erg;
END Lesen;
```

Vorname	Name	Matr.-Nr	4

Aufgabe 4: Vier-Gewinnt

Wir möchten ein Vier-Gewinnt-Spiel programmieren. Das Spielbrett steht aufrecht und besteht aus sieben Schächten, die jeweils sechs Steine aufnehmen können. Zwei Spieler spielen gegeneinander, der erste spielt mit schwarzen, der zweite mit roten Steinen. Jeder Spieler versucht, vier eigene Steine lückenlos in einer Zeile, einer Spalte oder in einer Diagonalen zu plazieren, bevor dies dem Gegner gelingt. Dazu werfen die Spieler jeweils abwechselnd einen Stein ihrer Farbe in einen der sieben Schächte. Wird ein Stein in einen Schacht eingeworfen, fällt er bis auf die tiefste noch freie Position dieses Schachtes.

4.1 (2 Pkt) Definieren Sie in Modula-3 eine geeignete Datenstruktur für das Spiel Vier-Gewinnt .
Hinweis: Die Prozeduren der Teilaufgaben 2) und 3) müssen auf der von Ihnen gewählten Datenstruktur operieren.

```
CONST Schaechte  = 7;
      Positionen = 6;

TYPE Inhalt    = {schwarz, rot, leer};
     Schacht   = [1..Schaechte];
     Position  = [1..Positionen];
     Spielbrett= ARRAY Schacht, Position OF Inhalt;
```

4.2 (1 Pkt) Deklarieren Sie eine Spielbrett-Variable und initialisieren Sie diese korrekt, damit das Spiel beginnen kann.

```
VAR sb :=
      Spielbrett{ARRAY Position OF Inhalt {Inhalt.leer, ..}, ..};
```

Vorname	Name	Matr.-Nr	5

4.3 (4 Pkt) Schreiben Sie eine Modula-3 Prozedur `SteinEinwerfen`, die als Parameter die Angabe des Schachtes und die Steinfarbe erhält. Kann der Stein eingeworfen werden, dann muß der Inhalt des Spielbretts entsprechend verändert sein.

```
PROCEDURE SteinEinwerfen(s: Schacht; i: Inhalt): BOOLEAN=
VAR p            : INTEGER;
    platzGefunden: BOOLEAN := FALSE;

BEGIN
  (* Suche freie Position im Schacht von unten nach oben *)
  p := 1;
  WHILE (p <= Positionen) AND (sb[s, p] # Inhalt.leer) DO
    INC(p)
  END;

  IF (p <= Positionen) THEN sb[s, p] := i; platzGefunden := TRUE; END;

  RETURN platzGefunden;
END SteinEinwerfen;
```

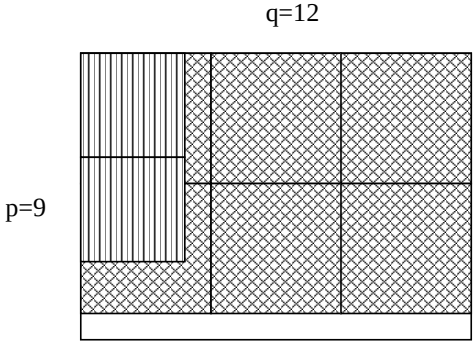
4.4 (6 Pkt) Schreiben Sie eine Modula-3 Funktion `HatGewonnen`, die ermittelt, ob ein Spieler durch Setzen des letzten Steins das Spiel gewonnen hat. Die Funktion erhält als Parameter die Schacht- und Positionsangabe des zuletzt eingeworfenen Steins.

```
PROCEDURE HatGewonnen(s: Schacht; p: Position): BOOLEAN=
VAR erg                : BOOLEAN := FALSE;
    zaehler, sNeu, pNeu: INTEGER;
    i                  : Inhalt;
BEGIN
  i := sb[s, p]; (* Farbe des Steins am untersuchten Platz *)
  (* Zaehle Steine auf den Geraden (-1,-1), (-1,0), (-1,1) und (0,-1) *)
  FOR xIndex := -1 TO 0 DO
    FOR yIndex := -1 TO 1 DO
      (* Pruefe nicht (0,0) und (0,1) *)
      IF (xIndex # 0) OR ((yIndex # 0) AND (yIndex # 1)) THEN
        zaehler := 1; (* Eingeworfener Stein *)
        (* Pruefe in positiver Richtung *)
        sNeu := s + xIndex; pNeu := p + yIndex;
        WHILE (zaehler < 4) AND GleicherInhalt(sNeu, pNeu, i) DO
          INC(zaehler); sNeu := sNeu + xIndex; pNeu := pNeu + yIndex;
        END;
        (* Pruefe in negativer Richtung *)
        sNeu := s - xIndex; pNeu := p - yIndex;
        WHILE (zaehler < 4) AND GleicherInhalt(sNeu, pNeu, i) DO
          INC(zaehler); sNeu := sNeu - xIndex; pNeu := pNeu - yIndex;
        END;
        IF (zaehler >= 4) THEN erg := TRUE; END; (* Vier gewinnt! *)
      END;
    END;
  END;
  RETURN erg;
END HatGewonnen;

PROCEDURE GleicherInhalt(s: INTEGER; p: INTEGER; i: Inhalt): BOOLEAN=
(* Prueft, ob Stein noch auf dem Spielbrett und die Farbe von i hat *)
VAR erg: BOOLEAN := FALSE;

BEGIN
  IF (s >= 1) AND (s <= Schaechte) THEN
    IF (p >= 1) AND (p <= Positionen) THEN
      IF (sb[s,p] = i) THEN erg := TRUE; END;
    END;
  END;
  RETURN erg;
END GleicherInhalt;
```

Vorname	Name	Matr.-Nr	6

Aufgabe 5: 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.

5.1 (6 Pkt) 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
      SIO.PutInt((qanz * panz) * (kmax *kmax)); SIO.Nl();
      RETURN (((qanz * panz) * (kmax *kmax) * 100) +
              BerechneKacheln(panz * kmax, qanz * kmax, kmax - 1));
    ELSE
      RETURN BerechneKacheln (p, q, kmax - 1); (* Probiere naechst
                                                  kleinere Kacheln! *)
    END;
  ELSE
    RETURN 0;
  END;
END BerechneKacheln;
```

Vorname	Name	Matr.-Nr	7

Aufgabe 6: Punkte und Rechtecke

Wir betrachten in dieser Aufgabe Punkte und achsenparallele Rechtecke in der Ebene. Ein Punkt ist definiert durch ein Koordinatenpaar (x,y). Ein achsenparalleles Rechteck ist definiert durch die linke, untere Ecke und die rechte, obere Ecke. Eine Ecke ist durch ein Koordinatenpaar (x,y) gegeben.

6.1 (6 Pkt) Geben Sie die Interface-Module für die abstrakten Datentypen Punkt und Rechteck an. Nehmen Sie alle die Operationen in die Schnittstellen der abstrakten Datentypen auf, die Sie in den nachfolgenden Teilaufgaben benötigen.

```
INTERFACE PunktADT;

TYPE Punkt <: REFANY;

PROCEDURE Erzeuge(x, y: INTEGER): Punkt;
PROCEDURE GetX   (p: Punkt): INTEGER;
PROCEDURE GetY   (p: Punkt): INTEGER;
PROCEDURE Verschiebe(p: Punkt; dx, dy: INTEGER);

END PunktADT.


INTERFACE RechteckADT;

IMPORT PunktADT;

TYPE Rechteck <: REFANY;

PROCEDURE Erzeuge(a, b: PunktADT.Punkt): Rechteck;
PROCEDURE GetA   (r: Rechteck): PunktADT.Punkt;
PROCEDURE GetB   (r: Rechteck): PunktADT.Punkt;
PROCEDURE Verschiebe(r: Rechteck; dx, dy: INTEGER);
PROCEDURE Skaliere  (r: Rechteck; dx, dy: INTEGER);
PROCEDURE Enthalten (r: Rechteck; p: PunktADT.Punkt): BOOLEAN;

END RechteckADT.
```

Vorname	Name	Matr.-Nr	8

6.2 (2 Pkt) Geben Sie den Deklarationsteil des Implementierungs-Moduls des ADTs `Rechteck` an (d.h. alles bis zur ersten implementierten Operation).

```
MODULE RechteckADT EXPORTS RechteckADT;

IMPORT PunktADT;

REVEAL Rechteck = BRANDED REF RECORD
                    linksunten: PunktADT.Punkt;
                    rechtsoben: PunktADT.Punkt;
                  END;

PROCEDURE Erzeuge(...
```

Gegeben seien eine Liste R von n achsenparallelen Rechtecken und eine Liste P von m Punkten.

6.3 (1 Pkt) Definieren Sie die Typen `ListeVonPunkten` und `ListeVonRechtecken`.

```
TYPE ListeVonPunkten    = ARRAY [1..m] OF PunktADT.Punkt;
     ListeVonRechtecken = ARRAY [1..n] OF RechteckADT.Rechteck;
```

Vorname	Name	Matr.-Nr	9

6.4 (5 Pkt) Schreiben Sie eine Modula-3 Funktion, die ermittelt, in welchem Rechteck der Liste R die meisten Punkte der Liste P liegen. (Liegen in mehreren Rechtecken gleich viele Punkte, so soll die Funktion eines davon zurückgeben)

```
PROCEDURE PunkteMax(lr: ListeVonRechtecken;
                    lp: ListeVonPunkten       ):
RechteckADT.Rechteck=

VAR zaehler: INTEGER;
    max    : INTEGER := 0;
    maxR   : RechteckADT.Rechteck := lr[1]; (* Initialisiere mit
                                               erstem Rechteck *)

BEGIN
  FOR i := FIRST(lr) TO LAST(lr) DO
    zaehler := 0;
    FOR j := FIRST(lp) TO LAST(lp) DO
      IF RechteckADT.Enthalten(lr[i], lp[j]) THEN
        INC(zaehler);
      END;
    END;
    IF (zaehler > max) THEN maxR := lr[i]; max := zaehler; END;
  END;

  RETURN maxR;
END PunkteMax;
```

Vorname	Name	Matr.-Nr

Diplom-Vorprüfung Informatik I
Teilprüfung Programmierung

5.3.2001, Prof. H. Lichter

Hinweise zur Bearbeitung:

- Schreiben Sie auf jedes Blatt **Vorname**, **Name** und **Matrikelnummer**
- Geben Sie Ihre Antworten in lesbarer und verständlicher Form an.
- 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	**8**	
Aufgabe 2	**11**	
Aufgabe 3	**8**	
Aufgabe 4	**16**	
Aufgabe 5	**12**	
Aufgabe 6	**5**	
SUMME	**60**	

Vorname	Name	Matr.-Nr

Aufgabe 1: Grammatik/Sprache

Betrachten Sie die Grammatik G = ({S, A, B}, {a, b}, P, S), wobei P gegeben ist durch:

S → aB	**A → a**	**B → b**
S → bA	**A → aS**	**B → bS**
	A → bAA	**B → aBB**

1.1 (4 Pkt) Geben Sie Syntaxdiagramme an, die die gleiche Sprache wie G beschreiben.

1.2 (4 Pkt) Bestimmen Sie L(G). Geben Sie eine umgangssprachliche, aber genaue Begründung an.

Vorname	Name	Matr.-Nr	3 Inf/L

Aufgabe 2: Rekursion und Iteration

Eine Eisenbahngesellschaft möchte mehrere neue Schienenstrecken verlegen. Ein Hersteller bietet Schienenstücke der Längen 1m, 2m und 5m an. Die Schienenstücke der Länge 1m haben den Preis p1, die Schienenstücke der Länge 2m haben den Preis p2 und die Schienenstücke der Länge 5m haben den Preis p5.

2.1 (8 Pkt) Schreiben Sie eine rekursive Modula3-Funktion
`MinPreis(laenge, p1, p2, p5:CARDINAL):CARDINAL`
die zu einer Strecke (gegeben durch eine natürliche Zahl, die die Länge der Strecke in Meter angibt) und den Preisen für die Schienenstücke den minimalen Preis für eine Überdeckung der Strecke mit den lieferbaren Schienenstücken berechnet.

Hinweis:
Sie können dabei die folgenden Funktionen zum Berechnen des Minimums verwenden:

```
MIN(X,Y  : CARDINAL): CARDINAL
MIN(X,Y,Z: CARDINAL): CARDINAL
```

Vorname	Name	Matr.-Nr	4 Inf/L

2.2 (3 Pkt) Geben Sie eine iterative Modula3-Prozedur an, die ebenfalls den minimalen Betrag berechnet. Dabei gelten jedoch die folgenden Bedingungen

1. p5 < 2.5 * p2 5m-Schienen sind 2m-Schienen vorzuziehen
2. p2 < 2 * p1 2m-Schienen sind 1m-Schienen vorzuziehen

Vorname	Name	Matr.-Nr

Aufgabe 3: Lisp

Gegeben ist die folgende Listenstruktur, die die Personaldaten eines Unternehmens speichert. Für jeden Mitarbeiter wird der Name, die Gehaltsart, das Eintrittsjahr und das monatliche Gehalt erfasst. Diese Listenstruktur sei in der freien Variablen `PersListe` gespeichert.

```
(((Peter Maier)    Lohn     1965 1200)
 ((Frank Mueller)  Gehalt   1989 1800)
 ((Pia Schmidt)    Lohn     1975 1450)
)
```

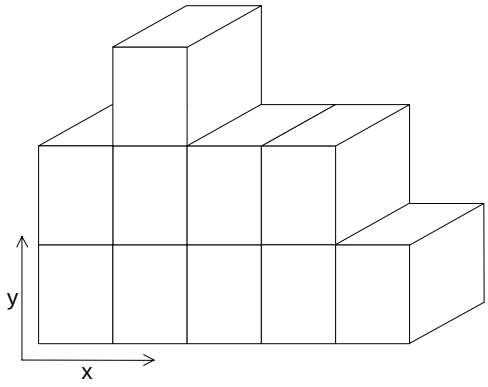
3.1 (4 Pkt) Schreiben Sie eine LISP-Funktion `einfuegen`, die als Parameter den Vornamen, den Namen, die Entlohnungsform (Lohn/Gehalt), das Eintrittsjahr und den momentanen Verdienst eines neuen Mitarbeiters erhält und die in `PersListe` gespeicherte Personalliste um einen entsprechenden Eintrag erweitert.

3.2 (4 Pkt) Es ist Monatsende und die Löhne und Gehälter sind fällig. Schreiben Sie zu diesem Zweck eine rekursive LISP-Funktion `summe`, welche die Gesamtsumme der durch das Unternehmen zu zahlenden Löhne und Gehälter aus der als Parameter übergebenen Personalliste ermittelt.

Vorname	Name	Matr.-Nr	6 Inf/L

Aufgabe 4: Zeiger und Listenstrukturen

In einem Containerhafen müssen die zu verladenden Container oft zwischengelagert werden (siehe Abbildung). Dazu können Container zu Stapeln **beliebiger** Höhe (y-Richtung) aufgetürmt werden. Um zusätzlich Platz zu sparen, werden die einzelnen Container-Stapel in einer Reihe (x-Richtung) zu einer Containerwand **beliebiger** Breite aneinander gestellt. Jeder Container ist gekennzeichnet durch eine eindeutige Nummer, sein Gewicht und den Namen des Eigentümers.



4.1 (3 Pkt) Entwerfen Sie geeignete Modula-3 Datenstrukturen, um den Aufbau einer solchen Containerwand im Rechner nachzubilden. Es muss möglich sein, ausgehend vom untersten Container des ersten Stapels (Ursprung des x-y-Achsenkreuzes in der Abbildung) einen beliebigen Container in der Containerwand zu erreichen.

Hinweis:
Entwerfen Sie die Datenstrukturen so, dass die nachfolgenden Teilaufgaben damit möglichst einfach implementiert werden können.

Vorname	Name	Matr.-Nr

Verwenden Sie im folgenden die von Ihnen in Teil 4.1) entworfenen Datenstrukturen:

4.2 (4 Pkt) Geben Sie eine Modula-3 Prozedur an, welche als Eingabe die eindeutige Nummer, das Gewicht und den Eigentümernamen eines neuen Containers bekommt und diesen auf dem Stapel x als obersten Container ablegt. Die Nummer des betreffenden Stapels x wird dazu ebenfalls als Parameter übergeben. Enthält die Containerwand weniger als x Stapel, so soll der Container als erster Container eines neuen Stapels an das rechte Ende der Containerwand angefügt werden.

Vorname	Name	Matr.-Nr	8 Inf/L

4.3 (9 Pkt) Um an einen auf einer tieferen Ebene gelagerten Container zu gelangen, müssen zunächst alle darüber liegenden Container auf den Nachbarstapel umgelagert werden. Da der Kran immer nur einen Container gleichzeitig transportieren kann, muss dies einzeln geschehen. Schreiben Sie eine Modula-3 Prozedur, welche alle Container oberhalb des Containers im Stapel x auf Ebene y, wie beschrieben, auf den rechts davon liegenden Stapel umlagert.

Beispiel: Umlagern aller Container oberhalb des Containers C = Position (2,1)

HINWEIS: Es kann vorausgesetzt werden, dass der Container mit der Position (x,y) immer vorhanden ist. Die Reihenfolge der Containerverlagerungen während der Operation spielt keine Rolle, entscheidend ist die Situation nach deren Abschluss. Es dürfen im Laufe der Operation jedoch keine Container gelöscht und/oder neu angelegt werden.

Vorname	Name	Matr.-Nr	9 Inf/L

Aufgabe 5: Datenabstraktion

In dieser Aufgabe soll eine Kontoverwaltung realisiert werden. Dazu stehen die abstrakten Datentypen `Buchung` und `Datum` zur Verfügung. Der Objektmodul `Konto` ist zu definieren und teilweise zu realisieren. Ein Konto besitzt neben einem Inhaber, einer Kontonummer und einer Bankleitzahl ebenfalls eine Buchungsliste, in der, nach Datum sortiert, alle Buchungen, die dieses Konto betreffen, gespeichert sind. Die Speicherung der Buchungen soll in einer einfach verketteten Liste erfolgen.
Die Schnittstellen der ADTen `Buchung` und `Datum` sind bereits vorgegeben.

```
INTERFACE Buchung;

IMPORT Datum;

TYPE T <: REFANY;

PROCEDURE NeueBuchung(d: Datum.T; b: REAL): T;
(* Erstellt eine neue Buchung mit dem
angegebenen Datum und Betrag (b) *)

PROCEDURE GetDatum(buchung: T): Datum.T;
(* Gibt das Datum der Buchung zurück *)

PROCEDURE GetBetrag(buchung: T): REAL;
(* Gibt den Betrag der Buchung zurück *)

PROCEDURE AlsText(buchung: T): TEXT;
(* Erzeugt eine textuelle Darstellung *)

END Buchung.
```

```
INTERFACE Datum;

TYPE Tag =   [1..31];
     Monat = [1..12];
     Jahr  = [1980..2100];

TYPE T <: REFANY;

PROCEDURE NeuesDatum(t: Tag; m: Monat;
                     j: Jahr): T;
(* Erzeugt ein neues Datum mit den Daten
für Tag, Monat und Jahr*)

PROCEDURE GetTag(datum: T): Tag;
(* Gibt des Tag des Datums zurück *)

PROCEDURE GetMonat(datum: T): Monat;
(* Gibt den Monat des Datums zurück *)

PROCEDURE GetJahr(datum: T): Jahr;
(* Gibt das Jahr des Datums zurück *)

END Datum.
```

5.1 (4 Pkt) Geben Sie die Schnittstelle zum Objektmodul `Konto` an. Deklarieren sie die geforderten Zugriffsprozeduren und fachlichen Funktionen. Insbesondere werden die Funktionen `BerechneSaldo` und `Kontoauszug` benötigt.

`BerechneSaldo` liefert den aktuellen Kontostand auf Basis der vorhandenen Buchungen.
`Kontoauszug` gibt alle Buchungen, die zwischen einem Anfangsdatum und einem Enddatum liegen, als Text zurück.

Vorname	Name	Matr.-Nr	10 Inf/L

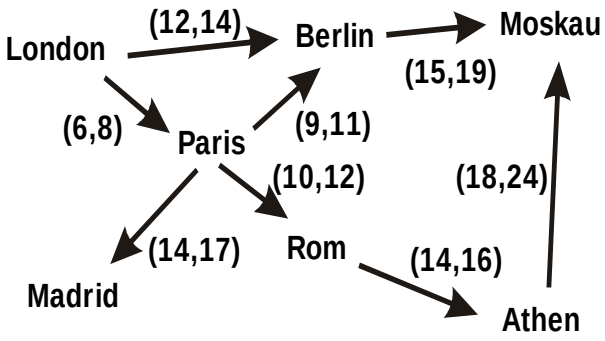
5.2 (2 Pkt) Geben Sie den Deklarationsteil (alle Typ- und Variablendeklarationen) im Implementierungs-Modul (Rumpf) des Objektmoduls „Konto“ an.

5.3 (6 Pkt) Implementieren Sie die Funktion `Kontoauszug` des Objektmoduls `Konto`. Erweitern Sie dazu das Interface `Datum` um geeignete Funktionen. Die Funktion `Kontoauszug` kann dann auf diese neuen Funktionen zurückgreifen.

Vorname	Name	Matr.-Nr	11 Inf/L

Aufgabe 6: Prolog

Die nebenstehende Grafik zeigt die Flugverbindungen zwischen verschiedenen Hauptstädten. Die Verbindungen sind unidirektional, das heißt, eine Flugverbindung existiert nur in Richtung der Pfeilspitze. Zu jedem Flug ist die Ablugzeit und die Ankunftszeit angegeben (alle Städte liegen in der gleichen Zeitzone).
Beispiel: Beim Flug London-Berlin ist die Abflugzeit in London 12.00 Uhr und die Ankunftszeit in Berlin 14.00 Uhr.



6.1 (2 Pkt) Man ist daran interessiert, ob eine Flugverbindung von einem Abflughafen zu einem Zielflughafen existiert. Lösen Sie das Problem mit PROLOG. Beschreiben Sie die vorhandenen Flugverbindungen und definieren Sie ein Prädikat `flug`, welches ermittelt, ob der Zielflughafen vom Startflughafen aus erreichbar ist.

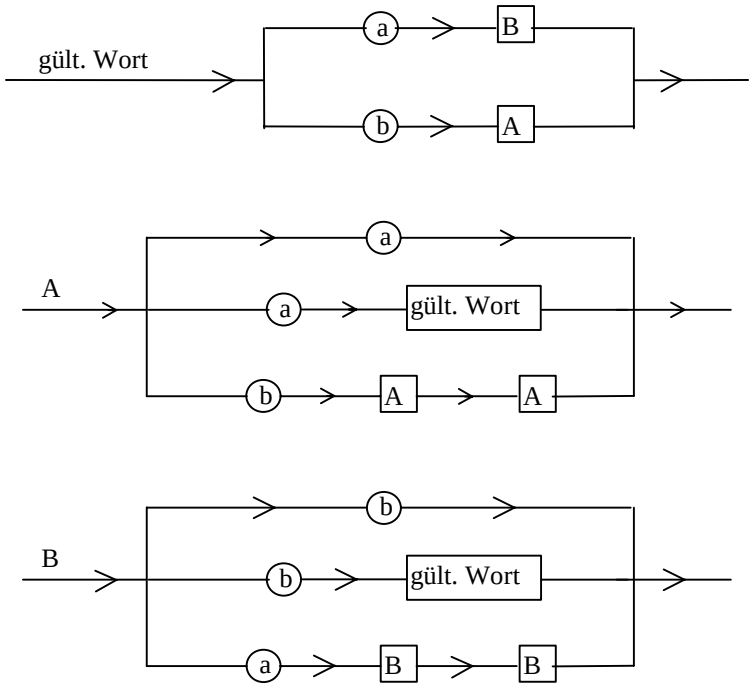
6.2 (3 Pkt) Definieren Sie auf Basis der vorhandenen Flugdaten ein Prädikat `flugzeit`, das ausgehend von einem Startflughafen die reine Flugdauer bis zur Landung am Zielflughafen berechnet. Dabei soll die folgende Bedingung gelten: Es dürfen nur solche Anschlussflüge auf dem Weg zum Zielflughafen genutzt werden, für die gilt, dass zwischen Landung und Weiterflug wenigstens eine und maximal zwei Stunden liegen.

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

Loesung INF

Aufgabe 1.1 (4 Pkt)

L(G) ist gleich der Sprache, die durch das Syntaxdiagramm für “gült. Wort“ beschrieben wird.



Aufgabe 1.2 (4 Pkt)

$L(G) = \{w \in \{a, b\}^* : |w| > 0, |w|_a = |w|_b \}$

Begündung:

Die Bedingung $|w| > 0$ muß stets erfüllt sein, da in jeder von S ausgehenden Produktionsfolge mindestens ein a oder ein b auftritt (da auf S nur die Produktionen 1) und 2) angewandt werden können).

Die Grammatik G besteht aus drei Teilen:
Teil 1: Die linke Seite der Produktionen ist S
Teil 2: Die linke Seite der Produktionen ist A.
Teil 3: Die linke Seite der Produktionen ist B.
Dabei sind die Struktur des zweiten und des dritten Teils gleich, jedoch sind A und B sowie a und b vertauscht.
Der erste Teil besteht aus zwei Regeln, für die das Gleiche gilt wie für den ersten und den zweiten Teil.

Weiterhin kann eine Produktionsfolge nur terminieren, wenn das Nichtterminal A durch a ersetzt und B durch b.
In einer mit A beginnenden Produktionfolge kann nur ein b erzeugt werden, wenn gleichzeitig zwei a's erzeugt werden. Um das Nichtterminal A (von S ausgehend) zu erhalten, muß jedoch zuvor ein b erzeugt werden. Aus A kann direkt (d.h. in einem Produktionsschritt) nur ein a erzeugt werden.
Das Analoge gilt für B (mit A und B bzw. a und b vertauscht).
Somit bedingt die Erzeugung eines a's die Erzeugung genau eines b's und umgekehrt. Für die Reihenfolge der a's und b's gibt es keine Bedingungen.

Aufgabe 2.1 (8 Pkt)

```
PROCEDURE MinPreis(laenge,pr1, pr2, pr5:CARDINAL):CARDINAL =
BEGIN

  (* Wenn die Laenge >= 5 ist, dann können für die erste Schiene alle drei
     Schienenarten verwendet werden.
     Der minimale Preis ist das Minimum aus allen drei Möglichkeiten, wobei
     sich der minimale Preis in allen drei Fällen durch rekursive Anwendung
     der Funktion berechnet. *)

  IF laenge >= 5 THEN RETURN MIN( pr5 + MinPreis(laenge-5,pr1,pr2,pr5),
                                  pr2+ MinPreis(laenge-2,pr1,pr2,pr5) ,
                                  pr1+ MinPreis(laenge-1,pr1,pr2,pr5));

  (* Wenn die Laenge >= 2 aber nicht >= 5 ist, dann gibt es nur zwei
     Moeglichkeiten fuer die erste Schiene. *)

  ELSIF laenge >= 2 THEN RETURN MIN(pr2+ MinPreis(laenge-2,pr1,pr2,pr5) ,
                                    pr1+ MinPreis(laenge-1,pr1,pr2,pr5));

  (* Fundierung der Rekursion *)

  ELSIF laenge = 1 THEN RETURN pr1;
  ELSIF laenge = 0 THEN RETURN 0;
  END;
END MinPreis;
```

Aufgabe 2.2 (3 Pkt)

```
PROCEDURE MinPreisItNb(laenge,pr1, pr2, pr5:CARDINAL):CARDINAL =
VAR rest5, rest2 :  CARDINAL;
BEGIN

  (* Vorgehen: Wähle so viele Schienen der Länge 5 wie möglich und danach
               so viele Schienen der Länge 2 wie möglich. Der Rest wird mit
               Schienen der Länge 1 aufgefüllt. *)

  rest5 := laenge MOD 5;
  rest2 := laenge MOD 2;
  RETURN  (pr5 * (laenge DIV 5) + pr2 * (rest5 DIV 2) + pr1 * rest2);
END MinPreisItNb;
```

Aufgabe 3.1 (4 Pkt)

```
;; Prozedur zur Einfuegen eines neuen Mitarbeiters
```

```
(defun einfuegen (vorname name lohnform eintritt verdienst)
   (setq PersListe (cons (list (list vorname name)
                               lohnform eintritt verdienst)
                         PersListe)
   )
)
```

Aufgabe 3.2 (4 Pkt)

```
;; Rekursive Funktion, welche die Gesamtsumme der zu zahlenden
;; Loehne und Gehaelter ermittelt

(defun summe (liste)
   (cond ((null liste) 0)
         (T            (+ (cadddr (car liste))
                          (summe  (cdr liste))
                       )
         )
   )
)
```

Aufgabe 4.1 (3 Pkt)

```
TYPE Container = RECORD
                   idnummer   : CARDINAL;
                   gewicht    : REAL;
                   eigentuemer: TEXT;
                 END;

     Stapelelement = REF RECORD
                           container: Container;

                           (* Verweis auf naechstes Element *)
                           nElement : Stapelelement;
                         END;

     Stapel = REF RECORD
                    stapel : Stapelelement;

                    (* Verweis auf naechsten Stapel *)
                    nStapel: Stapel;
                  END;
```

Aufgabe 4.2 (4 Pkt)

```
PROCEDURE EinlagernContainer(VAR ersterStapel: Stapel; stapelNr:
            CARDINAL; id: CARDINAL; gew: REAL; eigner: TEXT) =
VAR aktElement, neuesElement: Stapelelement;
    aktStapel: Stapel;
    x        : CARDINAL;

BEGIN
  (* Lege neues Stapelelement auf der Halde an *)
```

```
  neuesElement := NEW(Stapelelement,
                      container := Container{idnummer := id,
                                             gewicht := gew,
                                             eigentuemer := eigner},
                      nElement := NIL);

  IF ersterStapel = NIL THEN
    (* Ausnahme: Es gibt noch gar keinen Container in der Wand *)
    ersterStapel := NEW(Stapel, nStapel := NIL);
    ersterStapel^.stapel := neuesElement;
  ELSE
    (* Suche betreffenden Stapel bzw. Ende der Containerwand *)
    aktStapel := ersterStapel; x := 1;
    WHILE (x < stapelNr) AND (aktStapel^.nStapel # NIL) DO
      aktStapel := aktStapel^.nStapel; INC(x);
    END;

    (* Einlagern an der gefundenen Stelle *)
    IF x = stapelNr THEN
      (* Stapel x existiert bereits und enthaelt
         konstruktionsbedingt einen Container *)
      aktElement := aktStapel^.stapel;
      WHILE aktElement^.nElement # NIL DO
        aktElement := aktElement^.nElement;
      END;
      aktElement^.nElement := neuesElement;
    ELSE
      (* Container wird erster Container eines neuen Stapels *)
      aktStapel^.nStapel := NEW(Stapel, nStapel := NIL);
      aktStapel^.nStapel^.stapel := neuesElement;
    END;
  END;
END EinlagernContainer;
```

Aufgabe 4.3 (9 Pkt)

```
PROCEDURE UmlagernContainer(ersterStapel: Stapel; stapelNr:
                            CARDINAL; ebeneNr: CARDINAL)=
VAR aktStapel : Stapel;
    aktElement: Stapelelement;
    hilfsStapel, hilfsElement: Stapelelement;
    x, y: CARDINAL;
BEGIN
  (* Suche betreffenden Stapel mit Nummer stapelNr *)
  aktStapel := ersterStapel; x := 1;
  WHILE x < stapelNr DO
    aktStapel := aktStapel^.nStapel;
    INC(x);
  END;

  (* Durchlaufe kompletten Stapel, setze das y.te Element als
     letztes Element des Stapels und staple die weiteren in
     einen umgekehrten Hilfsstapel *)
  aktElement := aktStapel^.stapel; y := 1; hilfsStapel := NIL;
  WHILE aktElement # NIL DO

    (* Referenz auf Stapelelement fuer weitere Bearbeitung *)
    IF y >= ebeneNr THEN hilfsElement := aktElement; END;
```

```
    (* Setze aktuelles Stapelelement schon mal eins weiter *)
    aktElement := aktElement^.nElement;

    IF y = ebeneNr THEN
      (* Setze Nachfolger des zu befreienden Containers NIL *)
      hilfsElement^.nElement := NIL;
    ELSIF y > ebeneNr THEN
      (* Setze weitere Container an Anfang des Hilfsstapels *)
      hilfsElement^.nElement := hilfsStapel;
      hilfsStapel := hilfsElement;
    END;

    INC(y); (* Setze nun auch Nummer der Ebene eins hoch *)
  END;

  (* Lege den Hilfsstapel auf den rechten Nachbarstapel *)
  IF aktStapel^.nStapel = NIL THEN
    (* Rechter Nachbarstapel ist noch leer *)
    aktStapel^.nStapel := NEW(Stapel, nStapel := NIL);
    aktStapel^.nStapel^.stapel := hilfsStapel;
  ELSE
    aktStapel := aktStapel^.nStapel;
    IF aktStapel^.stapel = NIL THEN
      aktStapel^.stapel := hilfsStapel;
    ELSE
      aktElement := aktStapel^.stapel;
      WHILE aktElement^.nElement # NIL DO
        aktElement := aktElement^.nElement;
      END;
      aktElement^.nElement := hilfsStapel;
    END;
  END;
END UmlagernContainer;
```

Aufgabe 5.1 (4 Pkt)

```
INTERFACE Konto;

IMPORT Buchung;
IMPORT Datum;

PROCEDURE Init();
(* Initialisiert das Konto und löscht alle Buchungen *)

PROCEDURE GetInhaber(): TEXT;
(* Liefert den Kontoinhaber *)

PROCEDURE SetInhaber(n: TEXT);
(* Setzt den Inhaber *)

PROCEDURE GetKtoNummer(): TEXT;
(* Liefert die Kontonummer *)

PROCEDURE SetKtoNummer(nr: TEXT);
(* Setzt die Kontonummer *)
```

```
PROCEDURE GetBLZ(): TEXT;
(* Liefert die Bankleitzahl *)

PROCEDURE SetBLZ(b: TEXT);
(* Setzt die Bankleitzahl *)

PROCEDURE BerechneSaldo(): REAL;
(* Berechnet den Saldo und gibt diesen als REAL-Wert zurück *)

PROCEDURE BuchungAnhaengen(buchung: Buchung.T);
(* Fügt die übergebene Buchung hinten an die Buchungsliste an
*)

PROCEDURE BuchungLoeschen(buchung: Buchung.T);
(* Löscht die übergebene Buchung aus der Buchungsliste *)

PROCEDURE Kontoauszug(anfang: Datum.T; ende: Datum.T): TEXT;
(* Gibt alle Buchungen, deren Datum innerhalb der übergebenen
Intervallgrenzen liegen, als Text zurück *)

END Konto.
```

Aufgabe 5.2 (2 Punkte)

```
MODULE Konto;

IMPORT Buchung;
IMPORT Datum;

TYPE Buchungsliste = REF RECORD
                       buchung: Buchung.T;
                       next: Buchungsliste;
                     END;

VAR inhaber,ktonummer, blz: TEXT;
    buchungen: Buchungsliste;
```

Aufgabe 5.3 (6 Punkte)

```
INTERFACE Datum;

[...]

PROCEDURE Gleich(d1,d2: T): BOOLEAN;

PROCEDURE KleinerAls(d1, d2: T): BOOLEAN;

END Datum.


PROCEDURE Kontoauszug(anfang: Datum.T; ende: Datum.T): TEXT =
```

```
VAR aktuell: Buchungsliste;
    datum: Datum.T;
    auszug: TEXT;
BEGIN
  auszug := "";
  aktuell := buchungen;
  IF (aktuell # NIL) THEN
    datum := Buchung.GetDatum(aktuell^.buchung);
    WHILE    (aktuell  #  NIL)  AND  (Datum.KleinerAls(datum,
anfang)) DO
      aktuell := aktuell^.next;
      IF (aktuell # NIL) THEN
        datum := Buchung.GetDatum(aktuell^.buchung);
      END;
    END;
    WHILE (aktuell # NIL) AND (Datum.KleinerAls(datum, ende))
DO
      auszug  :=  auszug  &  (Buchung.AlsText(aktuell^.buchung)  &
"\n");
      aktuell := aktuell^.next;
      IF (aktuell # NIL) THEN
        datum := Buchung.GetDatum(aktuell^.buchung);
      END;
    END;
    IF Datum.Gleich(datum, ende) THEN
      auszug  :=  auszug  &  (Buchung.AlsText(aktuell^.buchung)  &
"\n");
    END;
  END;
  RETURN auszug;
END Kontoauszug;
```

Aufgabe 6.1 (2 Punkte)

```
d_flug(london, berlin, 12, 14).
d_flug(berlin, moskau, 15, 19).
d_flug(london, paris, 6, 8).
d_flug(paris, berlin, 9, 11).
d_flug(paris, madrid, 14, 17).
d_flug(paris, rom, 10, 12).
d_flug(rom, athen, 14, 16).
d_flug(athen, moskau, 18, 24).

flug(Von, Von).
flug(Von, Nach) :- d_flug(Von, Ueber, _, _),
                   flug(Ueber, Nach).
```

Aufgabe 6.2 (3 Punkte)

```
flugzeit(Von,Nach, Flugzeit) :-
          d_flug(Von,Ueber,Start, Ankunft),
          Dauer is Ankunft-Start,
          weiter_flug(Ueber,Nach,Ankunft,Restflugzeit),
          Flugzeit is Dauer+Restflugzeit.

weiter_flug(Von,Von,_,0).
weiter_flug(Von,Nach,Ankunft, Flugzeit) :-
       d_flug(Von,Ueber,StartWeiterflug,AnkunftWeiterflug),
       Diff is StartWeiterflug-Ankunft,
       Diff>=1,
       Diff=<2,
       Dauer is AnkunftWeiterflug-StartWeiterflug,
       weiter_flug(Ueber,Nach,AnkunftWeiterflug,Restflugzeit),
       Flugzeit is Dauer+Restflugzeit.
```