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

Testklausur

Ausgabe: Di, 30.01.2001

Lösung in der Globalübung: Di, 13.02.2001

Aufgabe 1 : Syntaxdiagramm und EBNF

Es sollen Bezeichner definiert werden, die folgenden Vorschriften genügen:

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

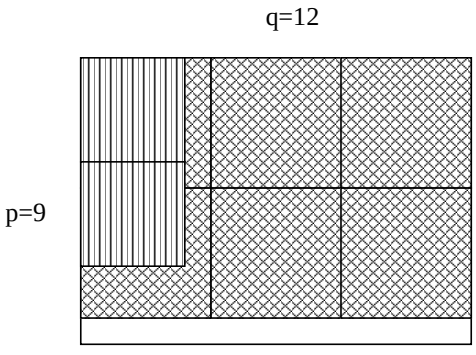
Die Produktionen für **Buchstabe** und **Ziffer** können als gegeben betrachtet werden.

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

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

Aufgabe 2: Kacheln

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



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

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

Aufgabe 3: Programmanalyse

Gegeben sei die folgende rekursive Prozedur:

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

Die Prozedur erzeugt also mit jeder Inkarnation keine oder eine Zeile Ausgabe.

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

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

#	p1	p2	Ausgabe (falls erzeugt)
1			
2			
3			
4			
5			
6			
7			

Aufgabe 4: Listen

Seien $x = (x_1,x_2,\ldots,x_m)$ und $y = (y_1,y_2, \ldots, y_n)$ zwei nicht-leere einfach verkettete Listen.
Aus diesen Listen soll eine neue Liste Z konstruiert werden, die folgenden Bedingungen genügt.

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

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

Aufgabe 5: Abstrakter Datentyp

Die Realisierung von Mengen in Modula-3 ist beschränkt auf Ordinaltypen als Elementtyp der Menge. Sollen andere Elementtypen in Mengen verwendet werden, so muß dafür eine geeignete Implementierung erstellt werden.
Entwerfen Sie einen Abstrakten Datentyp SetOfText, der eine Menge realisiert, deren Elementtyp der vordefinierte Typ TEXT ist. Als Operationen sollen bereitgestellt werden: Vereinigung, Schnitt, Aufnahme und Entfernen eines Elements aus der Menge sowie der Test des Enthaltenseins.

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

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

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

Aufgabe 6: Klassenhierarchie und Objekttypen

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

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

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

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

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

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

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

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

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

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

Aufgabe 7: Lisp

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

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

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

Programmierung **WS 2000 / 2001** **Prof. H Lichter**
Th. von der Maßen

Übung 1

Ausgabe: 27.10.00 Abgabe: 3.11.00 Besprechung: 8./9.11.00 in den Gruppen

Aufgabe 1.1: Definitionen

a) Geben Sie die Eigenschaften eines Algorithmus an und erläutern Sie diese.
b) Prüfen Sie die folgenden Aussagen auf ihre Richtigkeit und korrigieren Sie falsche Aussagen. Begründen Sie Ihre Korrektur.
   i. „Ein Algorithmus ist in einem endlichen oder unendlichen Text niedergelegt."
   ii. „Die Anzahl und die Ausführungszeit der Elementaroperationen eines Algorithmus´ ist beschränkt."
   iii. „Ein terminierender, deterministischer Algorithmus ist immer determiniert."
   iv. „Ein nicht-terminierender Algorithmus benötigt unendlich viel Speicherplatz."

(3 Punkte)

Aufgabe 1.2: Algorithmus

Sie stehen vor einem Getränkeautomaten (ähnlich dem vor Mensa V) und möchten eine Flasche Limonade erwerben. Formulieren Sie einen Algorithmus, welcher beschreibt, wie Sie die Flasche Limonade erhalten. Achten Sie dabei auf möglich auftretende Sonderfälle. Notieren Sie diesen Algorithmus umgangssprachlich unter Verwendung der vorgestellten Konstruktionsschemata für Algorithmen.

(6 Punkte)

Aufgabe 1.3: Grammatik

Gegeben sei die folgende Grammatik $G = (N, T, P, S)$, wobei

$N = \{ S, A \}$; $T = \{ 0, 1 \}$ und
P: $S \rightarrow 011 \mid A$
$A \rightarrow 0A11 \mid \varepsilon$

a) Geben Sie die Wörter an, die durch einen, vier und fünf Ableitungsschritte, gemäß der oben angegebenen Grammatik, erzeugt werden können. Geben Sie ebenfalls die einzelnen Ableitungsschritte an.
b) Geben Sie die durch die Grammatik erzeugte Sprache an.
c) Geben Sie eine äquivalente Grammatik *G´* an, welche außer dem Startsymbol keine zusätzlichen Nichtterminalsymbole benötigt.

(5 Punkte)

Aufgabe 1.4: Grammatik

Gegeben sei die folgende Sprache:

$$L = \{ w \in \{a,b\}^* \mid \#_a(w) = 3 \}$$

das heißt, es können Wörter beliebiger Länge erzeugt werden, welche die Buchstaben a und b enthalten und der Buchstabe a in einem Wort genau 3-mal vorkommt. Die folgenden Wörter sind beispielsweise in der Sprache enthalten:

abbbababb bbaaba aabbbba

Folgende Wörter sind <u>nicht</u> Bestandteil der Sprache:

abb abaabbba bbbb

a) Geben Sie das kürzeste Wort der oben angegebenen Sprache *L* an.
b) Geben Sie eine Grammatik *G* an, welche die Sprache *L* erzeugt.

(6 Punkte)

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

Übung 2

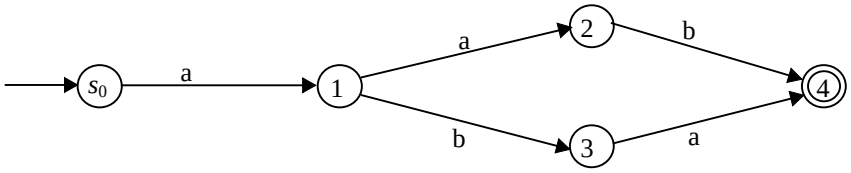
Ausgabe:	Di, 31.10.2000	**Abgabe in den Gruppen:**	Mi, 08.11.2000 Do, 09.11.2000	**Besprechung in den Gruppen:**	Mi, 15.11.2000 Do, 16.11.2000

Aufgabe 2.1: Syntax, Semantik *(3 + 2 +2 Punkte)*

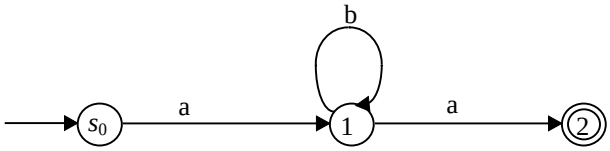
a) Was ist Syntax? Was ist Semantik? Erläutern Sie den Unterschied.
b) Impliziert gleiche Syntax auch gleiche Semantik? Geben Sie ein Beispiel.
c) Erläutern Sie folgende Aussage: Ein syntaktisch korrektes Programm ist nicht immer korrekt.

Aufgabe 2.2: Endliche Automaten *(4 Punkte)*

Ein endlicher Automat ist ein mathematisches Modell für einen einfachen Algorithmus, der endliche Zeichenfolgen (Wörter genannt) über einer gegebenen endlichen Zeichenmenge (Alphabet genannt) liest und diese entweder akzeptiert oder verwirft. Das Verhalten eines endlichen Automaten läßt sich durch ein Zustandsdiagramm beschreiben. Zustände werden als Kreise, Zustandsübergänge (Transitionen genannt) als Pfeile dargestellt. Den Zustand, in dem sich der Automat am Anfang befindet, nennt man s_0. Die anderen Zustände werden durchnumeriert. Es gibt einen oder mehrere Endzustände, die im Diagramm als doppelt umrandete Kreise dargestellt werden. An jeden Zustandsübergang wird geschrieben, welches Zeichen gelesen werden muß, damit der Übergang erfolgt. Man sagt, daß ein endlicher Automat genau dann eine Eingabe akzeptiert, wenn es für jedes gelesene Zeichen einen Übergang von dem aktuellen Zustand in einen Nachfolgezustand gibt und der Automat sich nach dem Lesen des letzten Zeichens in einem Endzustand befindet.
Beispiel: Der folgende endliche Automat akzeptiert genau die beiden Zeichenfolgen der aab und aba.



Es ist erlaubt, daß der Automat durch eine endliche Folge von Zustandsübergängen in einen bereits besuchten Zustand gelangt. Der folgende endliche Automat akzeptiert genau die Zeichenfolgen der aa, aba, abba, abbba, ...



Geben Sie das Zustandsdiagramm eines endlichen Auomaten über dem Alphabet {0, 1, 2, 3,4, 5, 6, 7, 8, 9} ∪ {E, X, P, +, -} an, der genau die Wörter akzeptiert, die Zahlen in der vom Taschenrechner bekannten EXP-Notation bilden.
Zum Beispiel sollen 123.456EXP-12, -987.654EXP+32 akzeptiert werden, nicht jedoch 012.345EXP-67, +23.456EXP-12 und -987.654EXP+3X.

Eine Zahl in der EXP-Notation hat ein „-“ als Vorzeichen oder kein Vorzeichen, danach folgt eine Zahl ohne führende Nullen vor dem Dezimalpunkt, dann der Dezimalpunkt und danach mindestens eine Stelle hinter dem Dezimalpunkt. Am Ende der Zahl steht „EXP” gefolgt von einem Exponenten. Der Exponent besitzt genau zwei Ziffern und hat ein „+“ oder ein „-“ als Vorzeichen.
Begünden Sie Ihre Antwort.

Aufgabe 2.3: EBNF, Syntaxdiagramme *(3 +3 + 3 Punkte)*

a) Um eine Datei eindeutig zu lokalisieren, wird ein Pfadname verwendet. Dieser besteht aus

- dem Buchstaben für das Dateisystem,
- einem Doppelpunkt,
- einem ersten \ für das Root-Verzeichnis,
- den Namen aller Verzeichnisse vom Root-Verzeichnis aus bis zu dem Unterverzeichnis mit der Datei, jeweils gefolgt von einem \,
- dem Dateinamensrumpf,
- einem Punkt und
- der Extension für den Dateityp

Zum Beispiel ist `c:\usr\pm3\bin\m3build.exe` ein Pfadname. Hiermit wird die Datei m3build.exe lokalisiert (wie sie nach der Installation von Modula-3 unter Windows vorliegt).
Beschreiben Sie die Syntax von Pfadnamen durch eine EBNF.
Erklären Sie Ihre Antwort.

b) Jede EBNF kann als Syntaxdiagramm formuliert werden.
Geben Sie (umgangssprachlich) einen Algorithmus an, der eine (beliebige) EBNF in ein Syntaxdiagramm überführt.
Erklären Sie Ihre Antwort.

c) Geben Sie ein Syntaxdiagramm für Adressen an, wobei eine Adresse ein Tripel ist, dessen erste Komponente ein Name (ein Paar betehend aus Vorname und Nachname), dessen zweite Komponente ein Paar bestehend aus einem Straßennamen und einer Zahl ohne führende Nullen, und dessen dritte Komponente ein Paar bestehend aus einer Postleitzahl und einem Ortsnamen ist.
Hierbei werden folgende (vereinfachende) Annahmen gemacht:

- Vornamen, Nachnamen, Straßennamen und Ortsnamen beginnen jeweils mit einem großen Buchstaben, die folgenden Buchstaben sind klein.
- Straßennamen enden stets mit ”strasse”.
- Ansonsten sind Vornamen, Nachnamen, Straßennamen und Ortsnamen beliebige endliche Wörter über dem Alphabet {a, b, c, ..., z}$\cup$ {A, B, C, ..., Z}.
- Postleitzahlen sind Wörter der Länge 5 über dem Alphabet {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}.

Erklären Sie Ihre Antwort.

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

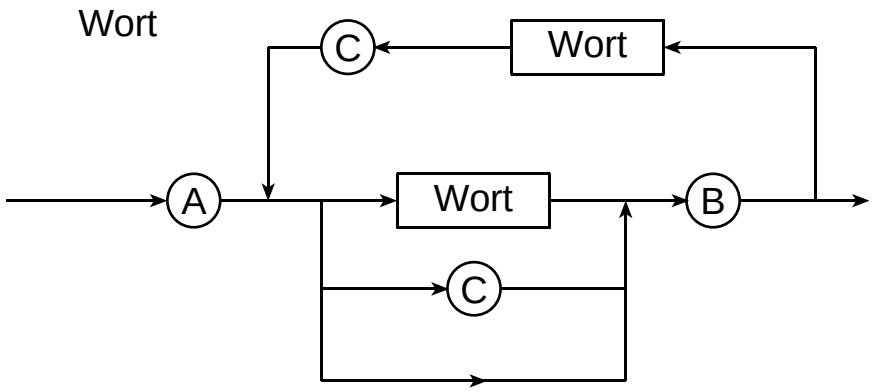
Übung 3

Ausgabe: Mi, 08.11.2000	**Abgabe in den Gruppen:** Mi, 15.11.2000 Do, 16.11.2000	**Besprechung in den Gruppen:** Mi, 22.11.2000 Do, 23.11.2000

WICHTIG: Alle Programme sind **sowohl** auf **Papier** in Ihrer Übungsgruppe **als auch** elektronisch mit Hilfe des **Online-Systems** abzugeben. Die Aufgaben 3.2 und 3.3 sind allein mit den Mitteln der funktionalen Programmierung zu lösen. Das heißt, es dürfen nur die Modula-3 Elemente verwendet werden, die im Abschnitt Funktionale Programmierung in der Vorlesung vorgestellt wurden. Lösungen, die andere Modula-3 Elemente wie Schleifen oder Variablen enthalten, werden nicht akzeptiert.

Aufgabe 3.1: Syntax *(5 Punkte)*

Gegeben sei das folgende Syntaxdiagramm, mit dem "Wörter" erzeugt werden können.



Prüfen Sie, ob die folgenden Wörter dem Syntaxdiagramm entsprechen. Geben Sie bei falschen Wörtern an, welche Zeichen mindestens gestrichen werden müssen (also möglichst wenige!), damit das Wort syntaktisch korrekt wird.

a) A B C
b) A A C B B
c) A B A A C B C B
d) A B A C C B A B C C B C B
e) A A C B A A B B C C B B
f) A B A C B A B A A C B B C B C C B C B
g) A C B A A A C B A A C B B C B B B C C B

Aufgabe 3.2: Modula-3 Programm für das Volumen einer Kugel *(8 Punkte)*

Entwickeln Sie ein Modula-3 Programm, das für einen gegebenen Kugeldurchmesser das Volumen der Kugel berechnet. Der Kugeldurchmesser wird am Programmanfang durch den Benutzer eingegeben.

a) Entwerfen Sie ein geeignetes Funktionsnetz (in grafischer Notation) für die Berechnung des Kugelvolumens, wie es auch in der Vorlesung beschrieben wurde. Verwenden Sie dazu nur die einfachen arithmetische Funktionen (+, *, -, /) sowie konstante Funktionsausdrücke.

b) Geben Sie den Funktionsausdruck zu dem in Teil a) entworfenen Funktionsnetz an.

c) Implementieren Sie das in Teil a) entwickelte Funktionsnetz als Modula-3 Programm, indem Sie für jede in Ihrem Funktionsnetz enthaltene Funktion eine entsprechende Modula-3 Funktion programmieren.

HINWEIS: Formel für das Volumen einer Kugel: $\frac{4}{3}\pi r^3$ mit r als Radius der Kugel

Aufgabe 3.3: Crazy Railways *(7 Punkte)*

Alle Güterzüge der Gesellschaft Crazy Railways müssen nach dem folgenden Gesetz zusammengestellt werden: Einen Zug der Länge a gibt es genau dann, wenn es auch einen Zug mit der Länge 3(a + 11) oder einen Zug mit der Länge a + 17 gibt. Natürlich hat ein Zug immer eine positive Länge.

Schreiben Sie ein Modula-3 Programm, das nach Eingabe der Länge des längsten Zuges die Länge des kürzesten Zuges berechnet. Erstellen Sie dazu eine **funktional-rekursive** Modula-3 Funktion mit der Signatur `KuerzesterZug(a: REAL): REAL`, wobei `a` die Länge des längsten Zuges und der Rückgabewert der Funktion die Länge des kürzesten Zuges sind. Erläutern Sie in wenigen Sätzen die von Ihnen gewählte Lösung.

Programmierung **WS 2000 / 2001** **Prof. H Lichter**
Th. von der Maßen

Übung 4

Ausgabe: 15.11.00 Abgabe: 22./23.11.00 Besprechung: 29./30.11.00 in den Gruppen

WICHTIG:
Alle Programme sind sowohl auf Papier in ihrer Übungsgruppe als auch elektronisch mit Hilfe des Online-Systems abzugeben. Die Aufgaben sind allein mit den Mitteln der funktionalen Programmierung zu lösen. Das heißt, es dürfen nur die Modula-3 Elemente verwendet werden, die im Abschnitt „Funktionale Programmierung“ in der Vorlesung vorgestellt wurden. Lösungen, die andere Modula-3 Elemente, wie Schleifen oder Variablen enthalten, werden nicht akzeptiert.

Aufgabe 4.1: Zeichenersetzung (3 Punkte)

Schreiben Sie eine rekursive Modula-3 Funktion „Substitution“, welche einen Text einliest und in diesem Text alle Vokale nach dem folgenden Schema ersetzt:

a → e	A → E
e → I	E → I
i → o	I → O
o → u	O → U
u → a	U → A

Der substituierte Text soll von der Funktion zurückgegeben und dann am Bildschirm ausgegeben werden. Nach dem Start des Programms soll der Benutzer einen beliebigen Text eingeben können, welcher an die Funktion „Substitution“ übergeben wird. Testen Sie Ihre Funktion mit einem geeigneten Hauptprogramm und mit dem folgenden Text:
„*Drei Chinesen mit dem Kontrabass, sitzen auf der Strasse und erzaehlen sich was. Dann kam die Polizei und was ist denn das? Drei Chinesen mit dem Kontrabass*.“

Aufgabe 4.2: Rekursive Funktion (4 Punkte)

Eine mathematische Funktion *f* sei wie folgt definiert:

$$f(n) = \begin{cases} 1, & falls \quad n = 1,2,3 \\ f(n-3) - 2 * f(n-2) + 3 * f(n-1), & falls \quad n \geq 4 \end{cases}$$

a) Geben Sie die Funktionswerte für n=1 bis n=10 an. Schreiben sie ebenfalls den Funktionsausdruck auf. Die Funktionswerte von *f(1)* bis einschließlich *f(n-3)* können direkt eingesetzt und brauchen nicht mehr weiter aufgelöst werden. Im folgenden ist der Funktionsausdruck für n=7 angegeben (die Funktionswerte bis *f(4)* können eingesetzt werden):

*f(7) = f(4)-2*f(5)+3*f(6)*
= 2-2(f(2)-2*f(3)+3*f(4))+3*(f(3)-2*f(4)+3*f(5))*
= 2-2(1-2*1+3*2)+3*(1-2*2+3*(f(2)-2*f(3)+3*f(4)))*
= 2-2(1-2*1+3*2)+3*(1-2*2+3*(1-2*1+3*2))*
= 28

Ein Ergebnis allein wird nicht gewertet.

b) Implementieren Sie eine rekursive Funktion „Berechne“ in Modula-3, welche die oben angegebene Funktion f implementiert. Testen Sie ihre Funktion mit einem geeigneten Hauptprogramm und berechnen Sie die Funktionswerte für n=1 bis n=25 und geben Sie diese an.

Aufgabe 4.3: Kubikwurzelberechnung (7 Punkte)

Die Kubikwurzel einer positiven reellen Zahl kann mit Hilfe des Newton-Verfahrens approximiert werden. Das Verfahren definiert eine Folge $(x_j)_{j\geq 0}$ mit $x_j \xrightarrow[j\to\infty]{} \sqrt[3]{x}$. Die Folge ist folgendermaßen definiert:

$$x_0 = x$$

$$x_j = \frac{x/(x_{j-1})^2 + 2x_{j-1}}{3} \qquad j > 0$$

Schreiben Sie eine Programm „Kubikwurzel“, welches eine Zahl einliest und die Folgenglieder $x_1, x_2, x_3, ..., x_k$ berechnet, bis $\left|x_k^3 - x\right|$ kleiner als eine vorgegeben Schranke von 0,0001 ist und x_k als Näherungswert für $\sqrt[3]{x}$ liefert.

a) Schreiben Sie eine Funktion `Abstand(y, x: REAL): REAL`, welche $\left|y^3 - x\right|$ berechnet.
b) Implementieren Sie eine Funktion `GutGenug(abstand, e: REAL): BOOLEAN`, welche testet, ob der Näherungswert innerhalb der Fehlertoleranzschranke liegt.
c) Implementieren Sie eine rekursive Funktion `BerechneKubikwurzel(x, y, fehler: REAL): REAL`, die einen weiteren Näherungswert berechnet, falls der aktuelle nicht gut genug ist. Ist der Wert gut genug, soll dieser zurückgegeben werden.
d) Schreiben sie ein geeignetes Hauptprogramm, um die von Ihnen implementierten Prozeduren und Funktionen zu testen. Lassen Sie Ihr Programm die Kubikwurzel aus folgenden Zahlen berechnen: 27 181 78,961 632,5 64

Aufgabe 4.4: Vermehrung (6 Punkte)

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. Wieviele 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.
Schreiben Sie eine rekursive Modula-3 Funktion `AnzahlLebewesen`, die n als Parameter erhält und die gesuchte Zahl liefert. Hinweis: Dazu ist es sinnvoll, eine eigene Funktion zu realisieren, welche die neugeborenen Lebewesen eines Jahres bestimmt.

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

Übung 5

Ausgabe:	Mi, 22.11.2000	**Abgabe in**	Mi,	29.11.2000	**Besprechung in**	Mi,	06.12.2000
		den Gruppen:	Do,	30.11.2000	**den Gruppen:**	Do,	07.12.2000

Aufgabe 5.1: Gültigkeitsbereich und Lebensdauer *(5 Punkte)*

Betrachten Sie folgendes Modula-3-Programm. Es besitzt Variablen und Prozeduren mit gleichen Namen. Modifizieren Sie den Programmtext, indem Sie die Variablen- und Prozedurnamen mit eindeutigen Indizes (z.B. a1 statt a) entsprechend ihrem Gültigkeitsbereich versehen. Zeichnen Sie anschließend das "Lebenslaufdiagrammm" (gemäß dem Beispiel aus der Vorlesung), welches die schreibenden und lesenden Zugriffe auf die Variablen sowie die Inkarnationen der Prozeduren zeigt, und ermitteln Sie damit alle Zwischen- und Endwerte der Variablen.

```
MODULE M EXPORTS Main;

VAR a, b, c: CARDINAL;

PROCEDURE P( a: CARDINAL; VAR b: CARDINAL) =
BEGIN
  a := a - b;
  b := b - c;
  c := c - a;
END P;

PROCEDURE Q ( a: CARDINAL; VAR b: CARDINAL)=

  PROCEDURE P( a: CARDINAL; VAR b: CARDINAL; VAR d: CARDINAL) =
  BEGIN
    c := c + b;
    a := a - c;
    b := b + a;
    d := c + a;
  END P;
VAR d : CARDINAL;
BEGIN
  a := a + b;
  P(a, b, d);
  b := b + a - d;
END Q;

BEGIN
  a := 1;  b := 2;  c := 3;  P( b, c);  Q( c, b);  P( c, a);
END M.
```

Aufgabe 5.2: Rekursion und Iteration *(3+3 Punkte)*

Statt Potenzen durch einfaches Multiplizieren zu berechnen, kann man auch den (zeitlich effizienteren) Weg der sukzessiven Quadratbildung wählen.
Zum Beispiel, statt b^8 mit b * b * b * b * b * b * b * b zu berechnen, kann man auch 3 Multiplikationen durchführen:

$$b^2 = b * b$$
$$b^4 = b^2 * b^2$$
$$b^8 = b^4 * b^4$$

Diese Methode der sukzessiven Quadratbildung läßt sich mit Hilfe der Regel

$$b^n = (b^2)^{n/2}\ ,\ \text{n gerade}$$

auch auf Potenzen übertragen, die keine Potenz von 2 sind.

a)
Schreiben Sie ein Modula-3-Programm für einen rekursiven Potenzierungsprozeß, welches ausschließlich sukzessive Quadratbildung verwendet.

b)
Schreiben Sie ein Modula-3-Programm für einen iterativen Potenzierungsprozeß, das ausschließlich sukzessive Quadratbildung verwendet.

Aufgabe 5.3: Iteration *(3 Punkte)*

Betrachten Sie folgende Berechnungsvorschrift (wobei a und b Parameter sind, welche noch nicht festgelegt sind):

$$f(0) = a$$
$$f(1) = b$$
$$f(n) = a\ f(n-1) + b\ f(n-2) \text{ für } n \geq 2$$

Schreiben Sie eine iterative Modula-3-Prozedur oder Funktion, welche zu einer beliebigen natürlichen Zahl n sowie Parametern a und b (Bestandteil der Eingabe) f(n) gemäß obiger Vorschrift berechnet.
Prüfen Sie die Prozedur oder Funktion mit einem Programm und den Parametern a=1, b=1 für $n \in \{1, ..., 10\}$.

Aufgabe 5.4: call-by-value, call-by-reference *(2+4 Punkte)*

a)
Was ist der Unterschied zwischen Wertparametern (”call-by-value”) und Referenzparametern (”call-by-reference”)? Welche Probleme können beim Gebrauch von Referenzparametern auftreten?

b)
Schreiben Sie eine Modula-3-Prozedur, die als Eingabe einen Satz erhält und folgende Werte an ihre Aufrufumgebung zurückliefert:

- die Anzahl der Kleinbuchstaben in dem Satz
- die Anzahl der Großbuchstaben in dem Satz
- die Anzahl der Konsonanten in dem Satz
- die Anzahl der Vokale in dem Satz
- die Anzahl der Doppellaute (ei, ai, au, Ei, Ai, Au) in dem Satz

Die Prozedur soll diese Werte in einem Durchlauf ermitteln.
Erläutern Sie im Kommentar den Zweck der gewählten Variablen.

Prüfen Sie die Prozedur mit einem Programm anhand der folgenden zwei Eingaben:

- *Eine korrekte Loesung rechnet hier zwanzig Vokale aus.*
- *Dieser Satz enhaelt einen Rechtschraibfehler.*

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

Übung 6

Ausgabe:	Di, 28.11.2000	**Abgabe in den Gruppen:**	Mi, 06.12.2000 Do, 07.12.2000	**Besprechung in den Gruppen:**	Mi, 13.12.2000 Do, 14.12.2000

Aufgabe 6.1: Matrizenmultiplikation *(4 Punkte)*

Schreiben Sie ein Modula-3 Programm, das eine gegebene quadratische (5 × 5)-Matrix A mit einer ebenfalls gegebenen quadratischen (5 × 5)-Matrix B multipliziert und die Ergebnismatrix C = A • B ausgibt.

Formel für die elementweise Matrizenmultiplikation:

$(A \bullet B)_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$, wobei wiederum eine (n × n)-Matrix entsteht.

a) Definieren Sie eine geeignete Modula-3 Datenstruktur, um eine (5 × 5)-Matrix mit REAL-Elementen zu speichern.
b) Implementieren Sie eine Modula-3 Funktion, die zwei gegebene (5 × 5)-Matrizen multipliziert und das Ergebnis zurückliefert. Verwenden Sie dabei die Datenstruktur aus Teil a).
c) Schreiben Sie ein Modula-3 Programm, das zwei (5 × 5)-Matrizen multipliziert und das Ergebnis wieder in Matrixform ausgibt. Weisen Sie den Ausgangsmatrizen die Werte mit Hilfe eines Feldaggregats zu, wie in der Vorlesung vorgestellt.

Aufgabe 6.2: Sortieren *(7 Punkte)*

Schreiben Sie ein Modula-3 Programm, das die Termine eines Arbeitstages aufsteigend beziehungsweise absteigend sortiert nach der Uhrzeit ausgibt.

a) Entwerfen Sie eine passende Datenstruktur, die maximal 10 Termine mit Beschreibung und Uhrzeit des Termins speichern kann.
b) Implementieren Sie eine Modula-3 Funktion, welche die übergebenen Termine nach der Uhrzeit sortiert zurückgibt.
c) Ergänzen Sie die in a) und b) entwickelten Teile zu einem Modula-3 Programm, das den Terminkalender auf Benutzerwunsch aufsteigend beziehungsweise absteigend sortiert nach der Uhrzeit ausgibt.

Aufgabe 6.3: Einwohnermeldeamt *(9 Punkte)*

Schreiben Sie ein Modula-3 Programm, das Einwohnermeldedaten verwaltet.

a) Entwerfen Sie eine sinnvolle Modula-3 Datenstruktur, welche die Meldedaten eines Einwohners speichert. Verwenden Sie dazu unter anderem Aufzählungstypen, Unterbereichstypen, Felder und Records. Folgende Daten müssen bei der Anmeldung erfasst werden:
   - der Familienname und maximal fünf Vornamen; der Rufname muss erkennbar sein.
   - das Geburtsdatum und der Geburtsort
   - das Geschlecht
   - die Wohnadresse
   - der Familienstand
   - die Namen des Ehepartners, falls der Einwohner verheiratet ist
   - das Datum der Anmeldung

b) Schreiben Sie eine Prozedur, die den Benutzer nach den Daten eines neuen Einwohners fragt und in der entworfenen Datenstruktur ablegt. Implementieren Sie geeignete Hilfsprozeduren für die verschiedenen Elemente der Datenstruktur. Die Plausibilität der Daten muss hierbei nicht geprüft werden.

c) Schreiben Sie eine Prozedur, welche die Daten aller Einwohner übersichtlich auf dem Bildschirm ausgibt. Implementieren Sie geeignete Hilfsprozeduren für die verschiedenen Elemente der Datenstruktur.

d) Ergänzen Sie die in a) - c) entwickelten Teile zu einem vollständigen Modula-3 Programm, das abhängig vom Benutzerwunsch einen neuen Einwohner erfasst oder die bereits erfassten Einwohner ausgibt. Das Programm muss maximal 10 Einwohner verwalten.

Programmierung	**WS 2000 / 2001**	**Prof. H. Lichter** **Th. von der Maßen**

Übung 7

Ausgabe: Mi, 06.12.2000	**Abgabe in den Gruppen:** Mi, 13.12.2000 Do, 14.12.2000	**Besprechung in den Gruppen:** Mi, 20.12.2000 Do, 21.12.2000

Aufgabe 7.1: Mengen (7 Punkte)

In dieser Aufgabe soll ein Modula-3 Programm implementiert werden, welches folgende Mengenrelationen realisiert: Gleichheit, Ungleichheit, Teilmenge, echte Teilmenge, Obermenge und echte Obermenge. Sie dürfen dazu die Modula-3 Mengenoperatoren verwenden, jedoch **nicht** die vordefinierten Mengenrelationen, mit Ausnahme der Enthalten-Sein-Relation. Stattdessen sollen alle Relationen durch Modula-3 Funktionen realisiert werden.

a) Die Elemente der Mengen sollen nur aus Kleinbuchstaben bestehen. Entwerfen Sie einen Modula-3 Datentyp `Buchstaben`, der eine Menge realisiert, welche Kleinbuchstaben aufnehmen kann.
b) Implementieren Sie die Relationen Gleichheit, Ungleichheit, Teilmenge, echte Teilmenge, Obermenge und echte Obermenge durch entsprechende Modula-3 Funktionen, welche jeweils zwei Mengen als Parameter übergeben bekommen.
   Bsp.: `PROCEDURE Gleich(menge1, menge2: Buchstaben): BOOLEAN =`
   Hinweis: Versuchen Sie die Funktionen möglichst geschickt aufeinander aufzubauen, um die Komplexität gering zu halten.
c) Realisieren Sie in Modula-3 ein Hauptprogramm, welches ermöglicht, zwei Mengen durch den Benutzer einzulesen. Im Anschluss soll der Benutzer aus einem Menü die oben realisierten Relationen auswählen können, die auf die beiden eingelesenen Mengen angewendet werden.

Aufgabe 7.2: Zeiger (4 Punkte)

a) Gegeben Sei das folgende Modula-3 Programm:

```
MODULE Zeiger EXPORTS Main;

TYPE CharRef = REF CHAR;

VAR x, y, z: CharRef;

PROCEDURE Mystery(a, b: CharRef) =
VAR z: CharRef;
BEGIN
  a^ := 'T';                    ❺
  a := b;                       ❻
  a^ := 'U';                    ❼
  z := NEW(CharRef);
  z := x;                       ❽
END Mystery;

BEGIN
  x := NEW(CharRef);
  y := NEW(CharRef);
  z := NEW(CharRef);            ❶
```

```
  x^ := 'A';
  y^ := 'B';
  z^ := 'C';                    ❷
  x^ := y^;                     ❸
  y^ := 'C';                    ❹
  Mystery(y, x);
  y := x;                       ❾
  x := NIL;                     ❿
END Zeiger.
```

Visualisieren Sie die Ablauf des Programms, in dem Sie die Zustände der Zeigervariablen nach jeder markierten Zuweisung in einer neuen Grafik darstellen. Geben Sie schließlich die Inhalte der Zeigervariablen x, y und z vor dem Beenden des Programms an.
Erläutern Sie zusätzlich textuell, was durch die Operationen an den Stellen ❶, ❷, ❸, ❻, und ❿ realisiert wird.

Aufgabe 7.3 Boxrangliste (9 Punkte)

Der Welt-Box-Verband WBV möchte eine Rangliste aller aktiven Boxer verwalten. Da immer wieder neuer Boxnachwuchs ausgebildet wird und in die Rangliste drängt und alte Boxer die aktive Laufbahn beenden, kann die Länge der Rangliste nicht auf eine feste Größe fixiert werden. Von jedem Boxer ist der Name, die Nation und das Gewicht bekannt.
Implementieren Sie die Rangliste in Modula-3 mittels einer einfach verketteten Liste (vom besten Boxer bis zum schlechtesten Boxer). Dabei sollen folgende Operationen durch geeignete interaktive Prozeduren realisiert werden:

- Aufnehmen eines neuen Boxers in die Rangliste (Dieser Boxer wird an das Ende der Rangliste gesetzt).
- Löschen eines Boxers aus der Rangliste.
- Aktualisieren der Rangliste nach einem Kampf. Dabei wird die folgende Regel zugrundegelegt: Der Boxer A wird in der Rangliste direkt hinter Boxer B platziert, falls A gegen B verliert und A vor dem Kampf vor B platziert war. Falls A ohnehin hinter B platziert war, ändert sich die Rangliste nicht.
- Ausgeben der aktuellen Rangliste auf dem Bildschirm.

Schreiben Sie ebenfalls ein geeignetes Modula-3 Hauptprogramm, welches eine Menü bereitstellt, aus dem die verschiedenen Operationen ausgewählt werden können. Testen Sie ihr Programm mit einer Reihe von Eingaben.

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

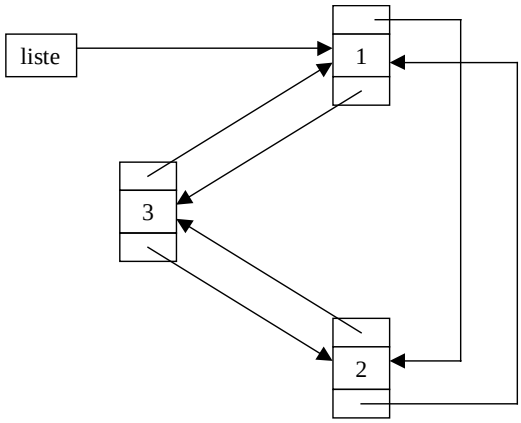
Übung 8

Ausgabe:	Di, 12.12.2000	**Abgabe in**	Mi,	20.12.2000	**Besprechung in**	Mi,	10.01.2001
		den Gruppen:	Do,	21.12.2000	**den Gruppen:**	Do,	11.01.2001

Vorbemerkung:
Zu jeder der folgenden Aufgaben ist ein geeignetes Hauptprogramm zu erstellen. Die Programme sind mit einer Reihe von Eingaben zu testen (und die Tests mit den Ergebnissen sind anzugeben).

Aufgabe 8.1: Doppelt verkettete, zyklische Liste *(1+2+5 Punkte)*

Bei einer doppelt verketteten Liste besitzt jeder Datensatz eine Vorwärts- und eine Rückwärtsreferenz. Eine zyklische Liste ist eine Liste, in der man von jedem Datensatz aus nach mindestens einem und höchstens endlich vielen Schritten wieder den Datensatz selber erreicht. Weiterhin hat in einer zyklischen Liste jeder Datensatz genau einen Nachfolger und bei einer doppelt verketteten, zyklischen Liste genau einen Vorgänger.
Um auf eine doppelt verkettete, zyklische Liste zugreifen zu können, ist ein Anker notwendig.
Ein Beispiel mit drei Datensätzen sieht wie folgt aus:



a) Entwerfen Sie eine Datenstruktur, mit der Sie eine nichtleere, doppelt verkettete, zyklische Liste aufbauen können, in welcher INTEGER-Werte abgelegt sind. Verwenden Sie Zeiger!

b) Schreiben Sie eine Prozedur, die den Aufbau einer solchen Liste ermöglicht.
Schreiben Sie weiterhin eine Prozedur, welche eine solche Liste auf dem Bildschirm ausgibt.

c) 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

Schreiben Sie eine Funktion Test, die die Kette prüft und TRUE liefert, wenn die Verkettung in Ordnung ist, sonst FALSE.

Konstruieren Sie im Rahmen Ihrer Tests Situationen, bei denen der Test fehlschlägt, d. h. die Verkettung beschädigt ist.

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

Aufgabe 8.2: Schülerliste *(2+5+5 Punkte)*

Ein Schüler wird identifiziert durch seinen Namen, seinen Vornamen, sein Alter und seinen Notendurchschnitt.

a) Entwerfen Sie eine Datenstruktur, mit der Sie eine einfach verkettete, lineare Liste von Schülern aufbauen können. Verwenden Sie Zeiger!
   Schreiben Sie weiterhin eine Prozedur, die den Aufbau einer solchen Liste ermöglicht sowie eine Prozedur, welche eine solche Liste auf dem Bildschirm ausgibt.

b) Schreiben Sie Prozeduren, die Folgendes leisten:
   Aus einer Liste von Schülern sollen diejenigen Schüler herausgefiltert werden, die über einem einzugebenden Alter liegen. Dabei soll eine neue Liste dieser Schüler produziert werden, welche nach Alter aufsteigend sortiert ist.
   Diese Liste soll anschließend folgendermaßen weiter bearbeitet werden: Aus der Liste sind diejenigen Schüler herauszufiltern, deren Durchschnitt über einer einzugebenden Grenze liegt. Die resultierende neue Liste soll nach Durchschnittsnote aufsteigend sortiert sein.
   Hinweise: Verwenden Sie das Konzept des Prozedurtyps. Die Schülerdaten der resultierenden Ergebnislisten sollen Kopien der Eingabeliste sein.

c) Realisieren Sie Prozeduren, die zu einer Schüler-Liste das arithmetische und das geometrische Mittel der Noten in der Liste berechnen.
   Schreiben Sie dazu eine Prozedur, welche auf die Noten einer Schüler-Liste sukzessive eine (einzugebende) zweistellige Funktion auf REAL-Werten anwendet und am Ende den entsprechenden (sich ergebenden) REAL-Wert zurückliefert. Verwenden Sie diese.

Hinweis:

Das arithmetische Mittel von n Zahlen a_1, ..., a_n ist definiert als $\frac{\sum_{i=1}^{n} a_i}{n}$.

Das geometrische Mittel von n Zahlen a_1, ..., a_n ist definiert als $\sqrt[n]{\prod_{i=1}^{n} a_i}$.

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

Übung 9

Ausgabe: Di, 19.12.2000	**Abgabe in den Gruppen:** Mi, 10.01.2001 Do, 11.01.2001	**Besprechung in den Gruppen:** Mi, 17.01.2001 Do, 18.01.2001

Aufgabe 9.1: Aufwandsabschätzung *(5 Punkte)*

Gegeben sind die folgenden Funktionen `EMPotenz` und `SQPotenz`, welche eine Potenz b^n durch einfaches Multiplizieren beziehungsweise sukzessive Quadratbildung (siehe auch Aufgabe 5.2) berechnen:

```
PROCEDURE EMPotenz(b: REAL;
            n: CARDINAL): REAL =
VAR erg: REAL := 1.0;

BEGIN
  FOR i := 1 TO n DO
    erg := erg * b;
  END;

  RETURN erg;
END EMPotenz;
```

```
PROCEDURE SQPotenz(b: REAL;
             n: CARDINAL): REAL =
VAR rekErg: REAL;

BEGIN
  IF n = 0 THEN
    RETURN 1.0;
  ELSIF n MOD 2 = 0 THEN
    rekErg := SQPotenz(b,n DIV 2);
    RETURN rekErg * rekErg;
  ELSE
    RETURN b * SQPotenz(b,n - 1);
  END;
END SQPotenz;
```

Schätzen Sie jeweils den Aufwand (Anzahl der notwendigen Multiplikationen) in Abhängigkeit des Exponenten n für beide Funktionen. Bestimmen Sie für die Funktion `SQPotenz` eine untere und eine obere Schranke für die Anzahl der benötigten Multiplikationen im günstigsten beziehungsweise schlechtesten Fall. Vergleichen Sie die Ergebnisse: Um welchen Faktor weniger Multiplikationen werden für n = 1024 (bzw. n = 1023) bei sukzessiver Quadratbildung gegenüber der einfachen Multiplikation benötigt?

HINWEIS: Die Operationen DIV und MOD seien ohne Multiplikation realisiert und tragen daher nichts zum Aufwand bei.

Aufgabe 9.2: Dateien *(6 Punkte)*

Erweitern Sie das Programm `Termine` aus der Musterlösung zu Aufgabe 6.2 so, dass die eingegebenen Termine in einer Datei gespeichert und später wieder gelesen werden. Implementieren Sie zu diesem Zweck geeignete Prozeduren, um die Daten des Terminkalenders einerseits in eine Datei zu schreiben und andererseits wieder zu lesen. Erweitern Sie anschließend das Programm so, dass es mit den implementierten Prozeduren zu Beginn die Termine aus einer lokalen Datei liest und am Ende des Programms wieder in dieser Datei speichert. Geben Sie das vollständige Programm und eine Datei mit mindestens drei bereits gespeicherten Terminen als Lösung ab.

HINWEIS: Verwenden Sie die Operationen des Moduls SIO, um die Termindaten zu speichern und zu lesen, indem Sie die Aus- bzw. Eingabe auf eine sequentielle (Text-)Datei, wie in der Globalübung vorgestellt, umstellen.

Aufgabe 9.3: Testen *(9 Punkte)*

Führen Sie einen systematischen Test des Programms aus Aufgabe 6.3 durch. Verwenden Sie dazu die speziell instrumentierte Implementierung des Programms `Meldeamt`, welche zusammen mit dem Übungsblatt auf den WWW-Seiten bereitgestellt wird.

a) Entwerfen Sie geeignete Blackbox-Testfälle, um die Eingabe eines Gemeldeten zu testen. Verwenden Sie als Spezifikation die Aufgabenstellung aus Übungsblatt 6 in Kombination mit folgenden Anforderungen:

- sämtliche Namen seien vom Typ TEXT und dürfen, wenn eine Eingabe zwingend ist, nicht leer sein
- ein Tagesdatum muss laut dem gregorianischen Kalender gültig sein, z.B. der 30.02.2000 ist kein gültiges Datum. Die Werte können jedoch beliebig, auch ohne führende Null, eingegeben werden, solange sie im Wertebereich 01.01.1800 bis 31.12.3000 liegen.
- Hausnummern sind auf den Bereich 1 bis 1000 beschränkt
- Postleitzahlen sind auf den Bereich 0 bis 99999 beschränkt
- ist der Familienstand verheiratet, fragt das Programm zusätzlich nach den Namen des Ehepartners, während bei Familienstand ledig, verwitwet oder geschieden nichts weiter passiert

Versuchen Sie Eingabeüberdeckung zu erreichen. Bestimmen Sie dazu sinnvolle Äquivalenzklassen für die einzugebenden Datenwerte und ermitteln Sie mit Hilfe einer einfachen Grenzwertanalyse dieser Äquivalenzklassen passende Testfälle. Definieren Sie zu jedem Testfall das Soll-Resultat. Nummerieren Sie sowohl die gefundenen Äquivalenzklassen als auch die Testfälle eindeutig und erstellen Sie eine Tabelle, welche für jeden Testfall die durch ihn überdeckten Äquivalenzklassen angibt.

HINWEIS: Nur die Dateneingabe soll getestet werden, die Menüsteuerung des Programms und die Datenausgabe (außer für Vergleichszwecke) brauchen nicht weiter berücksichtigt werden. Der Typ `TEXT` sei nicht begrenzt in seiner Größe.

b) Führen Sie die instrumentierte Version des Programms Meldeamt mit den in Teil a) entwickelten Testfällen aus und protokollieren Sie die Testergebnisse. Wo gibt es Abweichungen zwischen Soll- und Ist-Resultat? Wieviel Prozent der Zweige werden dabei überdeckt?

c) Zeichnen Sie das Flussdiagramm für die Prozedur `FamilienstandEingeben()` des Programms `Meldeamt`. Ermitteln Sie anschließend geeignete Testfälle, die für eine Zweigüberdeckung (C1 Test) der Prozedur notwendig sind. Geben Sie jeweils den erwarteten Rückgabewert der Prozedur als Soll-Resultat an.

d) Zeichnen Sie das Flussdiagramm für die Prozedur `PersonEingeben()` des Programms `Meldeamt`. Ermitteln Sie anschließend Testfälle, die für eine eingeschränkte Pfadüberdeckung der Prozedur notwendig sind. Geben Sie jeweils den erwarteten Rückgabewert der Prozedur als Soll-Resultat an.

Programmierung	WS 2000 / 2001	Prof. H. Lichter Th. von der Maßen

Übung 10

Ausgabe:	Di, 09.01.2001	**Abgabe in den Gruppen:**	Mi, 17.01.2001 Do, 18.01.2001	**Besprechung in den Gruppen:**	Mi, 24.01.2001 Do, 25.01.2001

Aufgabe 10.1: Objektmodul und ADT Schnittstellen (6 Punkte)

Implementieren Sie eine Diskographieverwaltung für CDs. Die Diskographie muss als Objektmodul implementiert werden. Die CDs sollen als abstrakter Datentyp realisiert werden. Eine CD enthält neben dem Interpreten und dem Titel eine Liste von Musikstücken (Tracks). Die Tracks sollen ebenfalls als abstrakter Datentyp realisiert werden.

a) Entwerfen Sie eine Schnittstelle für den abstrakten Datentyp „CD“. Dieser soll folgende Operationen bereitstellen:
   - Anlegen einer CD
   - Setzen des Interpreten
   - Lesen des Interpreten
   - Setzen des Titels
   - Lesen des Titels
   - Einfügen eines Tracks (an das Ende der Trackliste)
   - Löschen des letzten Tracks
   - Abfragen der Anzahl der Tracks
   - Anzeigen der Tracks, die auf der CD gespeichert sind (Namen und Dauer)
   - Abfragen der Gesamtspieldauer (Hinweis: Diese kann nicht gesetzt werden, sondern ergibt sich aus der Summe der Längen der einzelnen Tracks der CD)

b) Geben Sie eine Schnittstelle für den abstrakten Datentyp „Track“ an. Ein Track besteht aus einem Namen und der Länge in Sekunden. Geben Sie entsprechende Prozeduren und fachliche Funktionen an.

c) Geben Sie eine Modula-3 Schnittstelle des Objektmoduls „Diskographie“ an, welche Operationen bereitstellt, um eine CD hinzuzufügen und zu löschen sowie den Inhalt der Diskographie aufzulisten (d. h. die enthaltenen CDs anzuzeigen). Es sollen ebenfalls Testfunktionen bereitgestellt werden, die testen, ob eine Diskographie leer bzw. voll ist. Zusätzlich sollte eine Operation zur Initialisierung des Objektmoduls bereitgestellt werden.

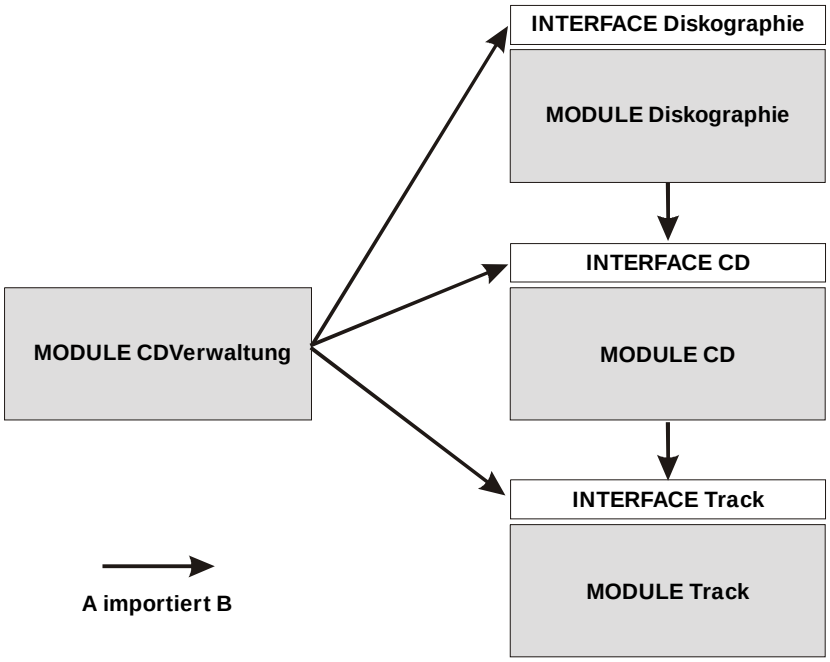
Aufgabe 10.2: Realisierung der Modulrümpfe (14 Punkte)

a) Implementieren Sie die in Aufgabe 10.1 angegebenen Schnittstellen in entsprechenden Modulen.
   Hinweise:
   - Eine Diskographie soll dabei maximal 5 CDs aufnehmen können. Hierzu kann ein Feld fester Länge genutzt werden.
   - Eine CD soll beliebig viele Tracks enthalten können. Dazu soll eine einfach-verkettete, lineare Liste realisiert werden.

b) Entwickeln Sie ein geeignetes Modula-3 Hauptprogramm, welches die folgenden Funktionen durch ein Menü bereitstellt:
   - Einfügen einer neuen CD. Dazu werden Titel und Interpret benötigt.

- Entfernen einer CD. Dazu sollen alle CDs mit ihrem Feld-Index aufgelistet werden. Durch Angabe einer Indexposition wird die CD spezifiziert, die aus der Diskographie gelöscht werden soll.
- Anzeigen aller in der Diskographie enthaltenen CDs mit Interpret, Titel, Gesamtspieldauer, Anzahl der Tracks sowie die enthaltenen Tracks mit Name und Länge.
- Hinzufügen eines oder mehrerer Tracks zu einer CD. Zuerst soll über die Indexposition angegeben werden, zu welcher CD Tracks hinzugefügt werden sollen. Nach jedem eingegebenen Track soll eine Abfrage erfolgen, ob ein weiterer Track hinzugefügt werden soll.
- Löschen des letzten Tracks einer CD. Dazu wird ebenfalls die Indexposition der CD benötigt. Auch hier soll eine Abfrage realisiert werden, ob ein weiterer Track gelöscht werden soll.
- Beenden des Programms

Das folgende Schaubild, soll noch einmal die Modulabhängigkeiten visualisieren:



Hinweis für die Online-Abgabe: Das Online-Formular ist aktualisiert worden. Tragen Sie die einzelnen Schnittstellen (Interfaces) und Module in die entsprechenden Felder des Online-Formulars ein.

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

Übung 11

Ausgabe:	Di, 16.01.2001	**Abgabe in den Gruppen:**	Mi, 24 .01.2001 Do, 25. 01.2001	**Besprechung in den Gruppen:**	Mi, 31.01.2001 Do, 01.02.2001

Aufgabe 11.1: Objektorientierung *(4 + 7 + 1 Punkte)*

In Übung 10 sollten Sie eine Diskographie realisieren. Diese sollen Sie nun wie folgt erweitern:

- Außer CDs soll die Diskographie auch Kassetten und Schallplatten enthalten. Diese Tonträgerarten sollen die in Übung 10 beschriebenen Merkmale und Funktionen besitzen (die Operationen „Löschen eines Tracks“ und „Angeben der Gesamtspielzeit“ brauchen nicht berücksichtigt zu werden).
- Zusätzlich sollen Tonträger eine Operation „PrintInfo“ zur Verfügung stellen, die neben sämtlicher verfügbarer Information zum Tonträger (also Titel, Interpret, alle enthaltene Tracks) auch die Tonträgerart (also CD, Schallplatte oder Kassette) als Text ausgibt.
- Tonträger sollen weiterhin in der Lage sein, sich selbst abzuspielen (das entspricht natürlich nicht der Realität). Dazu stellen Sie eine Operation „play“ zur Verfügung. Während CDs und Schallplatten direkt positioniert werden können (d.h. man kann eine Tracknummer beim Abspielen angeben), ist dies bei einer Kassette nicht möglich. Die Operation „play“ spielt dort den aktuell positionierten Track ab und stoppt dann.
- Für Kassetten soll eine einfache Operation zum Zurückspulen zur Verfügung gestellt werden. Diese soll die Kassette von der aktuellen Position (markiert durch einen Track) um genau einen Track zurückbewegen.

a) Entwerfen Sie in der in der Vorlesung angegebenen UML-Notation ein Klassendiagramm, das alle Klassen und alle Beziehungen zeigt, die Sie zur Lösung der Aufgabenstellung identifiziert haben. Kennzeichnen Sie abstrakte Klassen/ abstrakte Operationen und geben Sie pro Klasse alle Exemplarvariablen und alle öffentlichen Operationen an.
   Achten Sie darauf, gemeinsame Eigenschaften in entsprechenden Oberklassen anzulegen.
b) Implementieren Sie die unter a) entworfenen Klassen.
c) Schreiben Sie ein geeignetes Modula-3 Rahmenprogramm, und überprüfen Sie anhand einer Reihe von Eingaben Ihre Implementierung (und geben Sie die Ergebnisse Ihrer Testläufe an).

Hinweis:
Sollten Sie bereits in der 10. Übung mit der entsprechenden Aufgabe nicht zurechtgekommen sein, so können Sie die Musterlösung der 10. Übung, welche ab Freitag (19. 01. 2001) im Netz liegt, zur Orientierung verwenden.

Aufgabe 11.2: Objektmodul, Vertragsmodell *(8 Punkte)*

Entwerfen Sie ein Objektmodul, das die Datenstruktur zur Verwaltung einer Boxrangliste aus Aufgabe 9.2 kapselt. Definieren Sie zunächst die Schnittstelle des Objektmoduls, welche die folgenden Operationen anbieten muß:

1. Aufnehmen eines neuen Boxers (d.h., an das Ende der Rangliste setzen)
2. Streichen eines Boxers aus der Rangliste

3. Aktualisieren der Rangliste nach einem Kampf (Regel: Der Boxer A wird in der Rangliste direkt hinter dem Spieler B plaziert, wenn A gegen B verliert und A vor dem Kampf vor B platziert war. Falls A ohnehin hinter B platziert war, ändert sich die Rangliste nicht.)
4. Ausgeben der aktuellen Rangliste auf dem Bildschirm

Überlegen Sie sich Vor- und Nachbedingungen, die bei der Ausführung dieser Operationen gelten, und realisieren Sie diese mit Hilfe des in der Vorlesung vorgestellten Moduls Assertion in Ihrer Implementierung.
Erläutern Sie in den Kommentaren des Interfaces die von Ihnen definierten Vor- und Nachbedingungen.

Schreiben Sie ein geeignetes Modula-3 Rahmenprogramm, und überprüfen Sie damit anhand einer Reihe von Eingaben Ihre Implementierung (und geben Sie die Ergebnisse Ihrer Testläufe an).

Hinweise:
1. Bei der Behandlung der Vor- und Nachbedingungen ist unter Umständen die Definition von Testfunktionen hilfreich.
2. In den Programmbeispielen zur 9. Globalübung (Getränkeautomat) finden Sie das Modul Assertion.

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

Übung 12

Ausgabe:	Di, 23.01.2001	**Abgabe in den Gruppen:**	Mi, 31.01.2001 Do, 01.02.2001	**Besprechung in den Gruppen:**	Mi, 07.02.2001 Do, 08.02.2001

Aufgabe 12.1: Listen in LISP *(8 Punkte)*

Betrachten Sie die folgenden Listen 1 bis 5:

```
[1]     ((A B) C)
[2]     (A (((B C) D) E))
[3]     ((A B) ((C) (D E)))
[4]     ((E A C) (F (D B)))
[5]     ((C (((A) F) B) D) E)
```

a) Zeichnen Sie jeweils die Binärbaumdarstellung (siehe Vorlesung) für die Speicherstruktur der Listen 1 bis 5.

b) Geben Sie zu jeder der Listen 1 bis 5 eine entsprechende LISP-Form an, welche die Liste aus **einzelnen Atomen und der leeren Liste mittels CONS** aufbaut und dann einer Variablen (beispielsweise `liste1` bis `liste5`) zuweist.

c) Geben Sie zu jeder der Listen 1 bis 5 eine entsprechende LISP-Form an, welche die Atome A, B und C aus der ursprünglichen Liste extrahiert und als neue Liste (A B C) zurückliefert. Verwenden Sie dazu **nur die Funktionen CAR, CDR und CONS** sowie jeweils die in Teil b) definierte Variable (beispielsweise `liste1` bis `liste5`) - das heißt, die ursprüngliche Liste kann in Ihrer LISP-Form mehrfach verwendet werden.

Aufgabe 12.2: Numerik mit LISP *(6 Punkte)*

a) Definieren Sie eine Funktion `wurzel`, die für eine natürliche Zahl $a > 0$ eine natürliche Zahl b mit $b^2 = a$ zurückliefert. Falls keine solche Zahl existiert, soll die Funktion die Zahl 0 zurückgeben.

b) Definieren Sie eine Prozedur `pythagoras_tripel`, die zwei natürliche Zahlen a und b mit $0 < a < b$ als Eingabe erhält und drei natürliche Zahlen u, v und w, welche die Bedingung $w^2 = u^2 + v^2$ mit $a \le u \le b$ und $a \le v \le b$ erfüllen, als Liste zurückgibt. Falls ein solches Tripel (u, v, w) nicht existiert, soll die Prozedur die leere Liste zurückgeben.

Aufgabe 12.3: Nimmspiel in LISP *(6 Punkte)*

Das Nimmspiel ist ein Spiel für zwei Spieler und wird nach den folgenden Regeln gespielt:

> Ausgehend von einer vorher festzulegenden Anzahl Stäbchen nimmt abwechselnd jeder der beiden Spieler nach Gutdünken ein, zwei oder drei Stäbchen weg. Derjenige, der das letzte Stäbchen nehmen muss, hat verloren.

Überlegen Sie sich zunächst eine Strategie, mit der ein Spieler nach Möglichkeit gewinnt. Entwerfen Sie dann eine LISP-Prozedur `nimmspiel`, die als Eingabe die Anzahl der Stäbchen bei Spielbeginn erhält, anschließend das Nimmspiel interaktiv mittels `PRINT` und `READ` gegen einen menschlichen Benutzer spielt und dabei versucht zu gewinnen.

HINWEIS: Verwenden Sie die Funktion `(rem x y)`, um den Rest bei der Division zweier Zahlen x und y zu bestimmen.

Programmierung	**WS 2000 / 2001**	**Prof. H. Lichter** **Th. von der Maßen**

Übung 13

Ausgabe: Di, 30.01.2001	**Abgabe:** Mo, 12.02.2001	**Besprechung in den Gruppen:**	Mi, 14.02.2001 Do, 15.02.2001

Aufgabe 13.1: MAPCAR in LISP *(1 + 2 + 1 Punkte)*

a) Gegeben ist die folgende Funktion *mymapcar*:

```
(defun mymapcar (f liste)
  (defun iter (liste ergebnisliste)
    (cond ((null liste) ergebnisliste)
          (t (iter (cdr liste)
             (cons (funcall f (car liste)) ergebnisliste)))))
  (iter liste ()))
```

Die Funktion `mymapcar` erhält als Parameter eine Funktion und eine Liste. *mymapcar* wendet in einem iterativen Prozess die übergebene Funktion `f` auf alle Listenelemente der übergebenen Liste an und liefert die Ergebnisliste zurück. `funcall` bewirkt dabei, dass die Funktion `f` auf das erste Listenelement angewendet wird.
Beispiel: Eine gegebene Funktion *plus2*, `(defun plus2 (x)(+ x 2))` die zu einem übergebenen Parameter 2 addiert und das Ergebnis zurückliefert, kann bei *mymapcar* wie folgt verwendet werden: `(mymapcar #'plus2 '(1 2 3 4))` so dass als Antwortliste `(3 4 5 6)` erwartet wird. Leider ist jedoch dem Programmierer bei der obigen Implementierung von *mymapcar* ein Fehler unterlaufen. Erläutern Sie den Fehler, an welcher Stelle er auftritt und begründen Sie den Fehler.

b) Geben Sie eine korrekte Implementierung *mymapcar2* von *mymapcar* an.

c) Geben Sie nun einen Lambda-Ausdruck an, der das Quadrat einer Zahl berechnet und als Parameter für *mymapcar* in der Form `(mymapcar <lambda-Ausdruck> '(1 2 3 4))` verwendet werden kann.

Hinweis: Geben Sie zu allen Aufgaben, bei denen Sie PROLOG-Anfragen formulieren sollen, ebenfalls die Antwort(en) des PROLOG-Systems an.

Aufgabe 13.2: Fakten *(2 + 2 + 4 Punkte*)

a) Die unten angegebenen Tabellen geben darüber Auskunft, welche Vorlesungen von welchen Professoren gelesen werden sowie welche Studenten welche Vorlesungen besuchen. Geben Sie die Informationen der Tabellen als Faktenbasis in Prolog an. Definieren Sie hierzu die Relationen

- `studentHoertVorlesung`
- `profLiestVorlesung`

durch entsprechende Klauseln.

Student	Vorlesung
Balin	Kryptographie
Beutlin	Compilerbau
Boffin	Softwaretechnik
Brandybock	Mustererkennung
Pausbacken	Mustererkennung
Straffguertel	Compilerbau
Bolger	Datenkommunikation
Hornblaeser	Softwaretechnik
Stolzfuss	Datenkommunikation

Professor	Vorlesung
Hromkovic	Kryptographie
Indermark	Compilerbau
Nagl	Softwaretechnik
Spaniol	Datenkommunikation
Ney	Mustererkennung

b) Formulieren Sie unter Verwendung der Relationen aus Aufgabe 13.1 Regeln, die folgende Sachverhalte beschreiben:

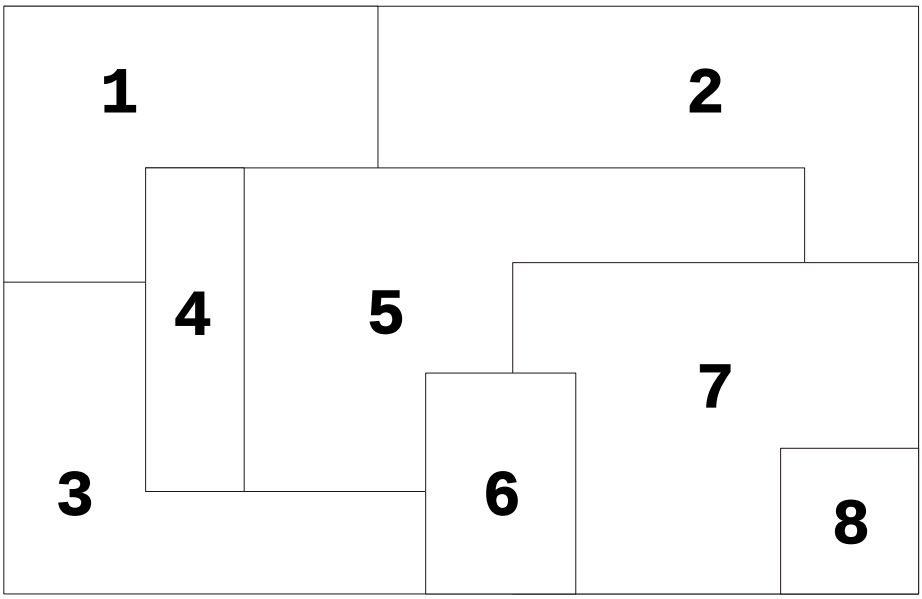
1. Balin hört die Vorlesung bei Professor Hromkovic, wenn Balin die Vorlesung Kryptographie hört und Kryptographie von Professor Hromkovic gelesen wird.
2. Ein Student hört die Vorlesung bei Professor Nagl, wenn...
3. Ein Student hört eine Vorlesung bei einem Professor, wenn...
4. Beutlin ist Kommilitone von Balin, wenn beide die gleiche Vorlesung hören.
5. Zwei Studenten sind Kommilitonen, wenn sie die gleiche Vorlesung hören.

c) Formulieren Sie unter Verwendung der obigen Fakten und Regeln folgende Anfragen in Prolog:

1. Hört Balin die Vorlesung „Compilerbau“?
2. Hört Balin die Vorlesung „Kryptographie“?
3. Liest Professor Nagl die Vorlesung „Softwaretechnik“?
4. Wird „Compilerbau“ von Professor Ney gehalten?
5. Welche Vorlesung besucht Brandybock?
6. Welche Vorlesung hält Professor Spaniol?
7. Wer liest die Vorlesung „Mustererkennung“?
8. Wer hört die Vorlesung „Softwaretechnik“?
9. Wer hört eine Vorlesung bei Professor Indermark?
10. Welche Studenten hören Vorlesungen von Professoren?
11. Ist Beutlin ein Kommilitone von Balin?
12. Ist Beutlin ein Kommilitone von Straffguertel?
13. Welche Kommilitonen hat Bolger?
14. Wer sind Kommilitonen?

Aufgabe 13.3: Landkarte färben (5 + *2 + 1 Punkte*)

Gegeben ist die folgende Landkarte mit acht Ländern:



a) Es stehen die Farben Rot, Gelb, Grün und Blau zur Verfügung. Jedes der Länder soll mit einer der vier Farben gefärbt werden, so dass aneinandergrenzende Länder unterschiedliche Farben besitzen. Lösen Sie das Problem mit Hilfe von Prolog, in dem Sie geeignete Fakten und Regeln formulieren. **Hinweis**: Definieren Sie Fakten, die ausdrücken, dass die vier verschiedenen Farben zur Verfügung stehen sowie wann zwei Farben verschieden sind. Legen Sie dann eine Regel fest, wann die obige Landkarte korrekt gefärbt ist.
b) Formulieren Sie eine Anfrage, die eine korrekte Färbung der Länder als Antwort ausgibt. Geben Sie eine korrekte Lösung zur Färbung der Landkarte an, die das PROLOG-System liefert.
c) Wieviele Lösungen zur Färbung der Landkarte gibt es insgesamt?