Übung 1

Ausgabe:	Mo, 19.10.98	**Abgabe:**	Mo, 26.10.98	**Globalübung:**	Di, 27.10.98
				Gruppen:	Do, 29.10.98
					Fr, 30.10.98

Verschiedene Zahlensysteme sowie insbesondere ihre Darstellung spielen eine wichtige Rolle in der Informatik; man denke dabei nur an die Binärdarstellung im Rechner. Zahlen werden heute vorrangig nach dem Prinzip des Dezimalsystems, einem Positionssystem, dargestellt. Dieses wurde im Mittelalter von Indien über den nahen Osten („arabische Ziffern“) zu uns gebracht. Es hat folgende Kennzeichen:

1) einheitliche **Basis g** aus der Menge der natürlichen Zahlen
2) unterschiedliche **Ziffern** für jede Zahl von **1 bis g-1**
3) spezielle **Ziffer für null**

Die Ziffern haben eine von ihrer Position abhängige **Gewichtung**: Die letzte (rechte) Ziffer hat das Gewicht 1, alle anderen Ziffern haben das g-fache Gewicht der rechts davon folgenden Ziffer. Eine Zahl k ($g^{n-1} <= k < g^n$) ist darstellbar durch n Ziffern.

Mit g =10 erhalten wir das **Dezimalsystem**, mit g=2 das **Dualsystem**.

Aufgabe 1.1: Zahlensysteme *(4 Punkte)*

Sie erhalten eine merkwürdige Nachricht bestehend aus den drei Zahlen:

"21 10 23"

Man sagt Ihnen, daß alle drei Zahlen gleich groß sind, allerdings in verschiedenen Zahlensystemen dargestellt. Um welche Zahlensysteme handelt es sich? Welches ist die kleinste mögliche Lösung?

Aufgabe 1.2: Binärbrüche *(4 Punkte)*

Im folgenden sind eine Reihe von Binärbrüchen dargestellt. Bestimmen Sie jeweils den entsprechenden Dezimalwert:

-1.0000 0.1 0.0001 -1.001 -0.101 1.1011 -0.1101 0.00011001100…

Aufgabe 1.3: Algorithmus *(6 Punkte)*

Sie stehen mit Ihrer Magnetkarte in der Hand vor einem Bankautomaten. Formulieren Sie in strukturierter Umgangssprache einen Algorithmus, der beschreibt, wie Sie nun Geld von Ihrem Konto abheben. Achten Sie dabei besonders auf mögliche Sonderfälle.

Aufgabe 1.4: Algorithmus *(6 Punkte)*

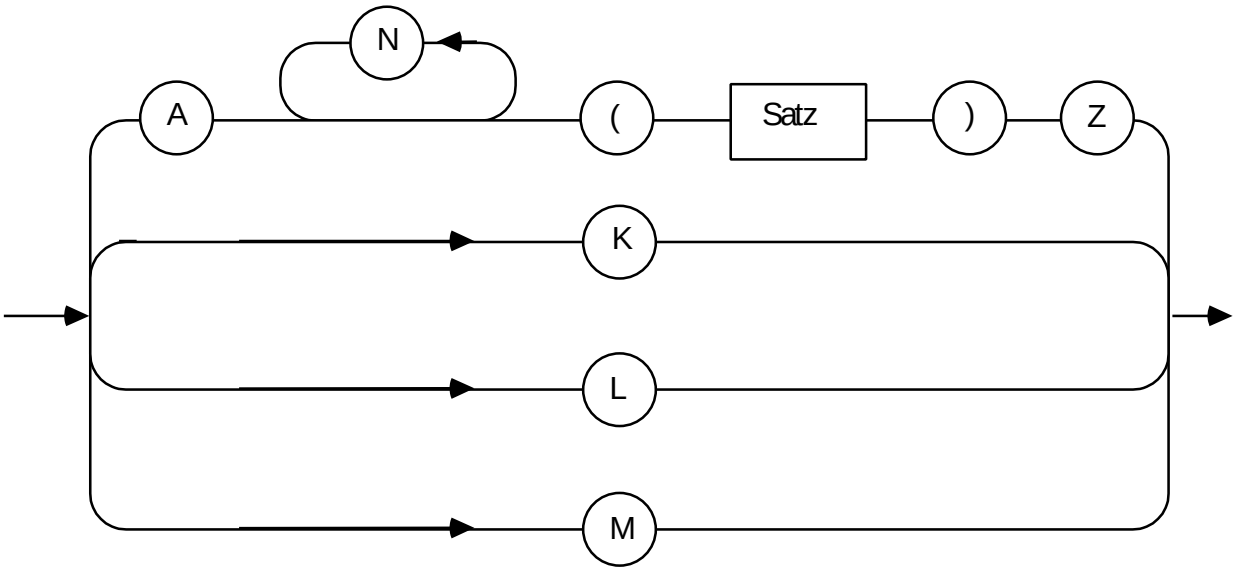
Sie stehen vor einem Regal mit einer Sammlung von Lexika und wollen einen Begriff nachschlagen. Formulieren Sie in strukturierter Umgangssprache einen Algorithmus, der beschreibt, wie sie den Begriff nachschlagen.

Übung 2

Ausgabe:	Mo, 26.10.98	**Abgabe:**	Mo, 02.11.98	**Globalübung:**	Di, 03.11.98
				Gruppen:	Do, 05.11.98
					Fr, 06.11.98

Aufgabe 2.1: Sprache *(6 Punkte)*

Gegeben sei das folgende Syntaxdiagramm mit dem "Sätze" erzeugt werden können:



Prüfen Sie die folgenden Sätze. Geben Sie bei den Sätzen an, die nicht dem Syntaxdiagramm entsprechen, welche Zeichen **mindestens gestrichen** werden müssen (also möglichst wenige!), damit der Satz syntaktisch korrekt wird, oder, ob eine solche Korrektur durch Streichungen nicht möglich ist.

a) A L Z
b) A (M) Z
c) (A N (A N (K) Z) Z)
d) A (N N N N (A (L) Z) Z
e) A N N K (A N N (A (N (K)) Z) Z) Z
f) Z (A N (N) Z) A

Aufgabe 2.2: Syntax von Telefonnummern *(4 Punkte)*

Geben Sie die Syntaxdiagramme für Telefonnummern (bezogen auf Aachen) an
a) für Ortsgespräche,
b) für Ferngespräche und
c) für Auslandsgespräche.

Aufgabe 2.3: Grammatik *(5 Punkte)*

Gegeben sei eine Sprache S über dem Alphabet A = { a, b, c, (,), [,] }. Für Wörter aus der Sprache S muß gelten, daß es für jede offene Klammer eine gleichartige schließende Klammer gibt, und daß die Klammern nicht verzahnt auftreten. Geben Sie eine Grammatik für die Sprache S an.

Beispiele:	Wörter aus S:	ε, abcbba, aaa(bbb(cc))a, aab(([acb]aaa)bb)
	Keine Wörter:	x, ab(bb(), ab(c[bb)c]ccc

Aufgabe 2.4: EBNF *(5 Punkte)*

Beschreiben Sie mit EBNF-Regeln die formale Sprache, die

a) aus den drei Funktionsnamen **`succ`**, **`pred`** und **`abs`** besteht,
b) aus allen Worten besteht, in denen nur abwechselnd **`a`** und **`b`** auftauchen,
c) aus allen Worten besteht, die gleichviele **`a`** und **`b`** enthalten,
d) aus allen (nicht leeren) Bezeichnern besteht, die aus Ziffern, Buchstaben und Unterstrich ('_') aufgebaut sind, aber nur mit einem Buchstaben beginnen,
e) alle ganzen Zahlen beschreibt.

Übung 3

Ausgabe: Mo, 02.11.98 **Abgabe:** Di, 10.11.98 **Globalübung:** Di, 10.11.98
Gruppen: Do, 12.11.98
Fr, 13.11.98

Aufgabe 3.1: Grammatik *(5 Punkte)*

Betrachten Sie die Grammatik G = {N, T, P, S} mit N = {S, B}, T = {a, b} und P wie folgt:

S $\rightarrow$ aSB | aB
aB $\rightarrow$ ab
bB $\rightarrow$ bb

a) Geben Sie den Strukturbaum für ein Wort (mind. Länge 5) der Sprache an.
b) Geben Sie die durch die Grammatik erzeugte Sprache an.
c) Formen Sie die kontextsensitive in eine äquivalente kontextfreie Grammatik um.

Aufgabe 3.2: Algorithmus *(5 Punkte)*

Geben Sie einen in strukturierter Umgangssprache (wie in der Vorlesung vorgestellt) formulierten Algorithmus an, der ganzzahlige, positive Dezimalzahlen in die römische Zahlendarstellung umrechnet. Für die römischen Zahlen gelte die übliche Darstellung mit den folgenden Ziffern:

Röm. Ziffer	I	V	X	L	C	D	M
Dezimalwert	1	5	10	50	100	500	1000

Beispiele:

Dezimal	1	2	3	4	5	6	7	8	9	10	11	12	15	21
Römisch	I	II	III	IIII	V	VI	VII	VIII	VIIII	X	XI	XII	XV	XXI

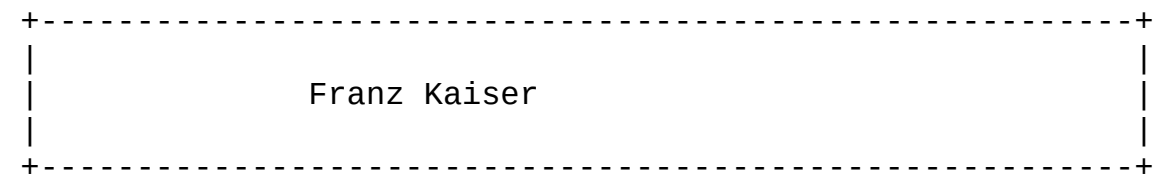
Hinweis: Die hier betrachteten römischen Zahlen sollen keine Abkürzungen, wie z. B. IX für die Dezimalzahl 9 oder MCM für die Dezimalzahl 1900 verwenden.

Aufgabe 3.3: Das erste und das zweite Modula-3 Programm *(10 Punkte)*

Schreiben Sie ein Modula-3 Programm,
a) das eine Visitenkarte (für im Programm fest vorgegebene Daten) in der folgenden Formatierung ausgibt:

```
+-----------------------------------------------------+
|                                                     |
|                    Franz Kaiser                     |
|                                                     |
|  Finkenstr. 17                    Tel.: 07518/00349 |
|  80005 Freimurg                   Fax.: 07518/00344 |
|                                                     |
+-----------------------------------------------------+
```

b) welches das folgende Namensschild für einen zu Beginn eingegebenen Namen erzeugt:



Hinweis: Der Name muß nicht in der Mitte des Schilds stehen, sondern darf an einem festen Punkt (zum Beispiel hier die Spaltenposition 16) beginnen.

Allgemeine Hinweise zu den Programmen in Modula-3:

Einführung in die Modula-3 Programmierumgebung: Globalübung, am 03.11.98

Die Programmieraufgabe 3.3 ist allein mit den **Mitteln der funktionalen Programmierung**, wie in der Vorlesung vorgestellt, zu bearbeiten. Die **Verwendung von Variablen/Konstanten sowie weiterer M3-Kontrollstrukturen ist nicht erlaubt** und wird als Lösung nicht anerkannt. Als **Lösung** abzugeben sind: Der Ausdruck des **Programmtexts** sowie der **Ausgabe einer Programmausführung** (auf Papier).

Für die passende Formatierung des Namensschilds in **Aufg. 3.3 b)** importieren Sie am besten das **Modul Text** aus der Modula-3 Standardbibliothek:

```
Copyright (C) 1994, Digital Equipment Corp.

A non-nil TEXT represents an immutable, zero-based sequence of characters. NIL
does not represent any sequence of characters, it will not be returned from any
procedure in this interface, and it is a checked runtime error to pass NIL to any
procedure in this interface.

INTERFACE Text;

IMPORT Word;

TYPE T = TEXT;

CONST Brand = "Text-1.0";

PROCEDURE Cat(t, u: T): T;
     Return the concatenation of t and u.

PROCEDURE Equal(t, u: T): BOOLEAN;
     Return TRUE if t and u have the same length and (case-sensitive) contents.

PROCEDURE GetChar(t: T; i: CARDINAL): CHAR;
     Return character i of t. It is a checked runtime error if i >= Length(t).

PROCEDURE Length(t: T): CARDINAL;
     Return the number of characters in t.

PROCEDURE Empty(t: T): BOOLEAN;
     Equivalent to Length(t) = 0.

PROCEDURE Sub(t: T; start: CARDINAL; length: CARDINAL := LAST(CARDINAL)): T;
    Return a sub-sequence of t: empty if start >= Length(t) or length = 0;
    otherwise the subsequence ranging from start to the minimum of start+length-1
    and Length(t)-1.

PROCEDURE SetChars(VAR a: ARRAY OF CHAR; t: T);
     For each i from 0 to MIN(LAST(a), Length(t)-1), set a[i] to GetChar(t, i).

PROCEDURE FromChar(ch: CHAR): T;
     Return a text containing the single character ch.

      ...  (* und so fort *)

END Text.
```

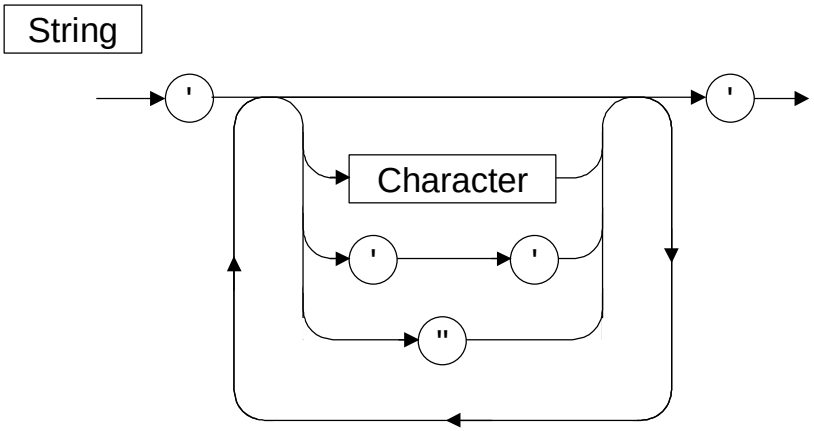
Übung 4

Ausgabe: Mo, 09.11.98 **Abgabe:** Di, 17.11.98 **Globalübung:** Di, 17.11.98
Gruppen: Do, 19.11.98
Fr, 20.11.98

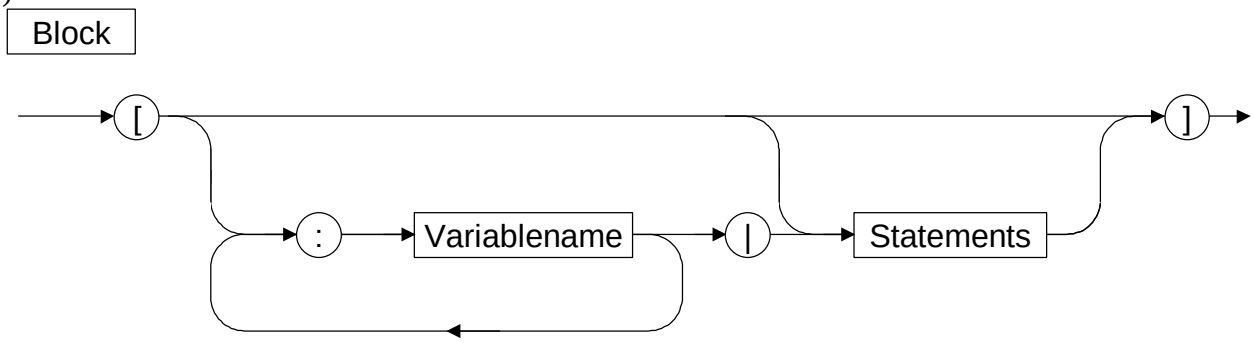
Aufgabe 4.1: Syntaxdiagramme und EBNF *(5 Punkte)*

Geben Sie für die folgenden Syntaxdiagramme äquivalente EBNF-Regeln an:

a)



b)



Aufgabe 4.2: Algorithmus für das Nimmspiel *(5 Punkte)*

Das Nimmspiel ist ein Spiel für zwei Spieler. Es 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.

Entwerfen Sie unter Verwendung der in der Vorlesung vorgestellten Konstruktionsschemata (d.h. sequentielle, alternative und iterative Ausführung von Algorithmen) einen Algorithmus, der das Nimmspiel auf einem Rechner so realisiert, daß ein menschlicher Benutzer gegen diesen Rechner spielen kann. Realisieren Sie in ihrem Algorithmus eine Strategie, mit welcher der Rechner nach Möglichkeit gewinnt.

Hinweis: Lösen Sie die folgenden Programmieraufgaben allein durch **Vernetzung von Funktionen**, wie in der Vorlesung beschrieben. **Die Deklaration von Variablen bzw. Konstanten** sowie die Verwendung von **Modula-3-Kontrollstrukturen** (mit **Ausnahme** von `IF ... THEN ... ELSE ...`) ist **nicht erlaubt**.

Als **Lösung** abzugeben sind: Der Ausdruck des **Programmtexts** sowie der **Ausgabe einer Programmausführung** (auf Papier).

Aufgabe 4.3: Einfacher Taschenrechner *(5 Punkte)*

Schreiben Sie ein Modula-3 Programm, das die vier Grundrechenarten (`+`, `-`, `*`, `/`) beherrscht. Dabei soll dem Benutzer zu Beginn die Möglichkeit gegeben werden, den arithmetischen Ausdruck einzugeben, der berechnet werden soll. Der Ausdruck muß in einer Zeile eingegeben werden und besteht aus dem ersten Operanden, einer `REAL`-Zahl, gefolgt vom Operator, ein Zeichen vom Typ `CHAR`, auf den schließlich der zweite Operand, wiederum eine `REAL`-Zahl, folgt. Das berechnete Resultat soll bei Abschluß des Programms ausgegeben werden.

Bsp.: Eingabe: `4.5+7.8` ⇒ Ergebnis: `12.3` Eingabe: `5.78-3.4` ⇒ Ergebnis: `2.38`
Eingabe: `6/5` ⇒ Ergebnis: `1.2` Eingabe: `3.4*7` ⇒ Ergebnis: `23.8`

Hinweis: Achten Sie besonders auf die Korrektheit der berechneten Resultate. Je nach Plattform (Unix/Windows) können sich hier Unterschiede ergeben!

Aufgabe 4.4: Gewicht einer Hohlkugel *(5 Punkte)*

Schreiben Sie ein Modula-3 Programm, welches das Gewicht einer Hohlkugel berechnet und ausgibt. Dabei soll das Programm dem Benutzer ermöglichen, sowohl die Radien (in cm) für die äußere bzw. innere Kugel als auch die spezifische Dichte (in g/cm^3) des Materials, aus dem der Kugelmantel gefertigt ist, einzugeben. Wird dabei ein kleinerer Radius für die äußere Kugel als für die innere Kugel eingegeben, soll der Wert 0 ausgegeben werden.

Verwenden Sie:

Volumen einer Kugel:	$4/3\pi r^3$
Gewicht einer Kugel :	Volumen * Dichte
Wert der Zahl Pi:	$\pi = 3.141592653$

Hinweise: Für die Lösung dieser Aufgabe sind **nur binäre Rechenoperationen** (mit `+`, `-`, `*`, `/`) zulässig. So ist zum Beispiel die Anweisung `a*b*b` nicht erlaubt, statt dessen verwende man `a*Quadrat(b)` oder `Mult(a,b)*b`. Versuchen Sie also die Berechnung in möglichst sinnvolle, einzeln zu implementierende Teilfunktionen zu zerlegen.

Für die Eingabe der Werte (Radien, Dichte) implementieren Sie eine Funktion:

```
PROCEDURE LeseWertEin(Frage: TEXT): REAL
```

Diese soll zunächst die im Parameter `Frage` enthaltene Zeichenfolge am Bildschirm ausgeben und die daraufhin vom Benutzer eingegebene `REAL`-Zahl zurückliefern. Beispiel für einen Funktionsaufruf:

```
LeseWertEin ("Geben Sie bitte den inneren Radius an: ")
```

Übung 5

Ausgabe: Mo, 16.11.98 **Abgabe:** Di, 24.11.98 **Globalübung:** Di, 24.11.98
Gruppen: Do, 26.11.98
Fr, 27.11.98

Hinweis: Bei allen Aufgaben dürfen **keine Variablen** verwendet werden und sind die protokollierten Resultate der entsprechenden Testläufe mit abzugeben!

Aufgabe 5.1: Rekursiver Primzahltest *(5 Punkte)*

a) Schreiben Sie eine rekursive Modula-3 Funktion, die prüft, ob eine Zahl (Typ `INTEGER`) eine Primzahl ist. Testen Sie Ihre Funktion in einem geeigneten Hauptprogramm für folgende Zahlen: 7, 12, 135, 277.

b) Zeichnen Sie das Funktionsnetz (gemäß Vorlesung) für die entwickelte Primzahltest-Funktion.

Aufgabe 5.2: Rekursive Modulo-Funktion *(5 Punkte)*

a) Schreiben Sie eine rekursive Modula-3 Funktion, welche die Modulo-Funktion (a mod b für $a \geq 0$ und $b \geq 0$) berechnet. Testen Sie Ihre Funktion in einem geeigneten Hauptprogramm für die Zahlenpaare: (10, 3); (19, 4); (121, 11); (129, 7).

b) Zeichnen Sie das Funktionsnetz (gemäß Vorlesung) für die entwickelte Modulo-Funktion.

Aufgabe 5.3: Umkehren eines Texts mittels Rekursion *(5 Punkte)*

Schreiben Sie eine rekursive Modula-3 Funktion, welche einen gegebenen Text (Typ `Text`) buchstabenweise umkehrt und diesen zurückliefert. Testen Sie Ihre Funktion in einem geeigneten Hauptprogramm anhand mindestens dreier verschiedener Texte. (Verwenden Sie bitte die Operationen aus dem Modul Text).

Aufgabe 5.4: Chiffrierung *(5 Punkte)*

Gegeben sei die folgende Chiffrierung:

1. Jeder Großbuchstabe des Alphabets wird durch den dritten Nachfolger ersetzt.
2. Jeder Kleinbuchstabe des Alphabets wird durch den fünften Nachfolger ersetzt.
3. Jede Ziffer wird durch den zweiten Vorgänger ersetzt.
4. Alle anderen Zeichen bleiben unverändert.
5. An den Grenzen des jeweiligen Bereichs (z.B. Großbuchstaben) findet eine Ersetzung durch das entsprechende Element am anderen Ende des Bereichs statt; beispielsweise ist der dritte Nachfolger des Buchstabens Y das Zeichen B, der zweite Vorgänger der Ziffer 1 ist die Ziffer 9.

a) Schreiben Sie eine Modula-3-Funktion, die ein Zeichen als Eingabe erhält und das nach obiger Vorschrift chiffrierte Zeichen ausgibt. Testen Sie Ihre Funktion in einem geeigneten Hauptprogramm für die folgenden Zeichen: b, K, z, Y, 5, 1, +.

b) Kombinieren Sie die Funktion aus Aufg. 5.3, die einen gegebenen Text umkehrt, mit der in Teil a) entwickelten Chiffrierfunktion. Schreiben Sie also ein Hauptprogramm, das einen vom Benutzer eingegebenen Text umgekehrt und chiffriert wieder ausgibt.

Schnittstelle des Moduls Text:

```
Copyright (C) 1994, Digital Equipment Corp.

A non-nil TEXT represents an immutable, zero-based sequence of characters. NIL
does not represent any sequence of characters, it will not be returned from
any procedure in this interface, and it is a checked runtime error to pass NIL
to any procedure in this interface.

INTERFACE Text;

IMPORT Word;

TYPE T = TEXT;

CONST Brand = "Text-1.0";

PROCEDURE Cat(t, u: T): T;
     Return the concatenation of t and u.

PROCEDURE Equal(t, u: T): BOOLEAN;
     Return TRUE if t and u have the same length and (case-sensitive) contents.

PROCEDURE GetChar(t: T; i: CARDINAL): CHAR;
     Return character i of t. It is a checked runtime error if i >= Length(t).

PROCEDURE Length(t: T): CARDINAL;
     Return the number of characters in t.

PROCEDURE Empty(t: T): BOOLEAN;
     Equivalent to Length(t) = 0.

PROCEDURE Sub(t: T; start: CARDINAL;
  length: CARDINAL := LAST(CARDINAL)): T;
      Return a sub-sequence of t: empty if start >= Length(t) or length = 0;
      otherwise the subsequence ranging from start to the
      minimum of start+length-1 and Length(t)-1.

PROCEDURE SetChars(VAR a: ARRAY OF CHAR; t: T);
     For each i from 0 to MIN(LAST(a), Length(t)-1), set a[i] to GetChar(t, i).

PROCEDURE FromChar(ch: CHAR): T;
     Return a text containing the single character ch.

PROCEDURE FromChars(READONLY a: ARRAY OF CHAR): T;
     Return a text containing the characters of a.

PROCEDURE Hash(t: T): Word.T;
     Return a hash function of the contents of t.

PROCEDURE Compare(t1, t2: T): [-1..1];
     Return -1 if t1 occurs before t2, 0 if Equal(t1, t2), +1 if t1 occurs after
     t2 in lexicographic order.

PROCEDURE FindChar(t: T; c: CHAR; start := 0): INTEGER;
     If c = t[i] for some i in [start~..~Length(t)-1], return the smallest such
     i; otherwise, return -1.

PROCEDURE FindCharR(t: T; c: CHAR;  start := LAST(INTEGER)): INTEGER;
     If c = t[i] for some i in [0~..~MIN(start, Length(t)-1)], return the
     largest such i; otherwise, return -1.

END Text.
```

Übung 6

Ausgabe: Mo, 23.11.98 **Abgabe:** Di, 01.12.98 **Globalübung:** Di, 01.12.98
Gruppen: Do, 03.12.98
Fr, 04.12.98

Aufgabe 6.1: Gültigkeitsbereich und Lebensdauer *(6 Punkte)*

Das folgende Programm besitzt Variablen und Prozeduren mit gleichen Namen. Modifizieren Sie den Programmtext, indem Sie die Bezeichner mit eindeutigen Indizes (zum Beispiel `a1` statt `a`) entsprechend ihrem Gültigkeitsbereich versehen. Zeichnen Sie anschließend das Ablaufdiagramm mit der Lebensdauer der Variablen (gemäß dem Beispiel aus der Vorlesung) und ermitteln Sie damit alle Zwischen- und Endwerte der Variablen.

```
MODULE M EXPORTS Main;

VAR a, b, c: INTEGER;

PROCEDURE P( a: INTEGER; VAR b: INTEGER)=
BEGIN
  a := a + b;
  b := b + c;
  c := c + a
END P;

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

  PROCEDURE P( a: INTEGER; VAR b: INTEGER)=
  BEGIN
    c := c + b;
    a := a + c;
    b := b + a
  END P;

BEGIN
  a := a + b;
  b := b + a - c;
  P( b, c)
END Q;

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

Aufgabe 6.2: Flagge Alphanumericas *(5 Punkte)*

Die vereinigten Staaten von Alphanumerica haben im Zuge der Automatisierung ihrer Flaggenindustrie einen Wettbewerb für die eleganteste Programmierung ihrer Flagge ausgeschrieben. Diese hat folgendes Aussehen (für die Größe k=8 mit Kommentaren):

```
--------********  1. Zeile: k Minuszeichen gefolgt von k Sternsymbolen
----****----****  2. Zeile: Zweimal (k/2 Minuszeichen, k/2 Sternsymbole)
--**--**--**--**  3. Zeile: Viermal (k/4 Minuszeichen, k/4 Sternsymbole)
-*-*-*-*-*-*-*-*  4. Zeile: Achtmal (k/8 Minuszeichen, k/8 Sternsymbole)
```

Die Flaggen gibt es in den Größen 2, 4, 8, 16, 32. Schreiben Sie ein Modula-3 Programm, das die Größe der Flagge abfragt und die zugehörige Flagge ausgibt.

Aufgabe 6.3: Schleifenkonversion *(3 Punkte)*

In dem unten angegebenen Programm fehlen die Rümpfe der Prozeduren `MitRepeat`, `MitWhile` und `MitLoop`. Ergänzen Sie die fehlenden Rümpfe so, daß jede dieser Prozeduren eine zu der Prozedur `MitFor` äquivalente Berechnung unter Verwendung der jeweiligen Wiederholungsanweisung ausführt. Der Aufruf des Programms soll also zu jeder zulässigen Eingabe viermal die gleiche Ausgabe erzeugen.

```
MODULE Fakult EXPORTS Main;

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

VAR eingabe: CARDINAL;     (* Variable fuer den Eingabewert *)

PROCEDURE MitFor (n: CARDINAL): CARDINAL =
(* Berechnet Fakultaet der Zahl n mit einer FOR-Schleife *)
VAR fakultaet: CARDINAL;   (* Rueckgabewert der Funktion *)

BEGIN
  fakultaet := 1;          (* Spezialfall n=1 *)
  FOR i := 2 TO n DO
    fakultaet := fakultaet * i;
  END;
  RETURN fakultaet;
END MitFor;

PROCEDURE MitWhile (n: CARDINAL): CARDINAL =
(* Umsetzung mit einer WHILE-Schleife *)
BEGIN
END MitWhile;

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

PROCEDURE MitLoop (n: CARDINAL): CARDINAL =
(* Umsetzung mit einer LOOP-Schleife *)
BEGIN
END MitLoop;

BEGIN
  (* Eingabe des Benutzers erfragen und Loesung berechnen. *)
  SIO.PutText("Von welcher Zahl wuenschen Sie die Fakultaet zu wissen? ");
  eingabe := SIO.GetInt();
  SIO.PutText("Mit FOR   : "); SIO.PutInt(MitFor(eingabe));    SIO.Nl();
  SIO.PutText("Mit REPEAT: "); SIO.PutInt(MitRepeat(eingabe)); SIO.Nl();
  SIO.PutText("Mit WHILE : "); SIO.PutInt(MitWhile(eingabe));  SIO.Nl();
  SIO.PutText("Mit LOOP  : "); SIO.PutInt(MitLoop(eingabe));   SIO.Nl();
END Fakult.
```

Aufgabe 6.4: Implementierung römischer Zahlen *(6 Punkte)*

Entwickeln Sie einen Algorithmus für die Umrechnung positiver, ganzer Zahlen in die römische Zahlendarstellung (wie in Aufg. 3.2 beschrieben), der maximal zwei Schleifenanweisungen enthält. Implementieren Sie diesen als Modula-3 Programm.

HINWEIS: Globale Variablen und Variablenparameter sind nicht erlaubt!

Übung 7

Ausgabe:	Mo, 30.11.98	**Abgabe:**	Di, 08.12.98	**Globalübung:**	Di, 08.12.98
				Gruppen:	Do, 10.12.98
					Fr, 11.12.98

HINWEIS: In der Globalübung, am Di 01.12.98, werden einige Grundregeln zum Thema "Guter Modula-3 Programmierstil" vorgestellt.

Aufgabe 7.1: Datenstrukturen *(8 Punkte)*

Entwerfen Sie je eine Modula-3 Datenstruktur für
a) das Datum mit Uhrzeit,
b) die Daten einer Person (Adresse, Geburtsdatum, Familienstand, ...),
c) ein Bankkonto,
d) die Daten eines Angestellten,
e) einen Einkaufszettel und
f) den Bücherbestand einer Bibliothek.

Sie sollten bei der Definition komplexerer Datenstrukturen auf bereits definierten Strukturen aufbauen. Überlegen sie sich hierzu sinnvolle Unterdatenstrukturen.

Aufgabe 7.2: Lottomanisches Lotto *(6 Punkte)*

Im Land Lottomania ist es aufgrund der großen Nachfrage notwendig geworden, das staatliche Lotto zu automatisieren. Das lottomanische Lotto hat folgende Form:

> Ein Tip ist eine Zahl aus dem Bereich 1 bis 49. Ein Lottospiel besteht nun darin, daß der Spieler einen bis maximal sechs Tips abgeben darf. Je nachdem wieviele dieser Tips mit den später tatsächlich gezogenen Zahlen aus 1 bis 49 übereinstimmen, errechnet sich der Gewinn des Spielers. Mit **einem** Lottozettel kann ein Spieler maximal zehn Spiele gleichzeitig machen.

a) Entwerfen Sie eine geeignete Modula-3 Datenstruktur, um einen solchen Lottozettel zu speichern.
b) Implementieren Sie in Modula-3 Prozeduren, die das Ausfüllen und Ausgeben am Bildschirm eines solchen Lottozettels erlauben. Testen Sie diese mit Hilfe eines geeigneten Hauptprogramms.

Aufgabe 7.3: Bruchrechnen *(6 Punkte)*

Ein Bruch b läßt sich mit Hilfe eines Arrays d folgendermaßen, in seiner Dezimalform mit k Nachkommastellen, exakt darstellen:

$$b = \sum_{i=1}^{k} d[i] * 10^{-i}$$

Entwerfen Sie eine passende Modula-3 Datenstruktur für diese Bruchdarstellung und implementieren Sie damit ein Modula-3 Programm, das sämtliche negativen Potenzen (das heißt diejenigen mit Exponent -1, -2, ...) der Zahl 5, die sich mit maximal 10 Nachkommastellen darstellen lassen, berechnet und ausgibt.

HINWEIS: Implementieren Sie hierzu eine Funktion, die das Resultat einer Division eines solchen Bruchs mit einer positiven, ganzen Zahl berechnet. Implementieren Sie außerdem entsprechende Funktionen/Prozeduren für die Initialisierung und Ausgabe eines solchen Bruchs in Dezimalform.

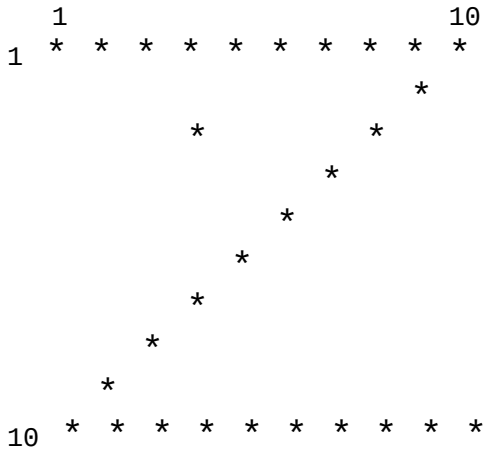
Programmierung **WS 1998/99** **Prof. H. Lichter**
M. Schnizler

Übung 8

Ausgabe:	Mo, 07.12.98	**Abgabe:**	Di, 15.12.98	**Globalübung:**	Di, 15.12.98
				Gruppen:	Do, 17.12.98
					Fr, 18.12.98

Aufgabe 8.1: Matrix *(7 Punkte)*

Gegeben ist die folgende 10 x 10 Matrix mit Elementen vom Typ `CHAR`:



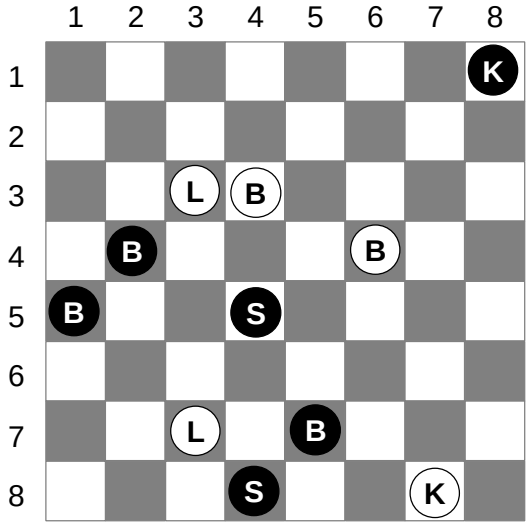
a) Entwerfen Sie eine Modula-3 Datenstruktur, um eine solche Matrix zu speichern.

b) Implementieren Sie ein Modula-3 Programm, welches zunächst eine Variable des in Teil a) entworfenen Typs mit dem oben dargestellten Muster initialisiert und anschließend den Benutzer fragt, welche Operation (repräsentiert durch folgende Zeichen vom Typ `CHAR`) auf diese Matrix angewendet werden soll:

   `v` = Spiegeln der Elemente an der vertikalen Symmetrieachse
   `h` = Spiegeln der Elemente an der horizontalen Symmetrieachse
   `d` = Spiegeln der Elemente an der Hauptdiagonalen
   `n` = Spiegeln der Elemente an der Nebendiagonalen
   `i` = Drehen der Elemente um 90^o im Uhrzeigersinn
   `g` = Drehen der Elemente um 90^o gegen den Uhrzeigersinn

   Daraufhin wird die jeweils entsprechend umgeformte Matrix ausgegeben. Dies geschieht solange, bis der Benutzer das Programm durch Eingabe des Zeichens `e` für Ende abbricht.

Aufgabe 8.2: Schach *(13 Punkte)*



Gegegeben sei die obige Stellung auf einem Schachbrett mit Bauern (B), Läufern (L), Springern (S) und Königen (K) der jeweiligen Farbe:

a) Entwickeln Sie einen Modula-3 Datentyp für ein Schachbrett, der geeignet ist, beliebige Stellungen abzuspeichern. Das heißt, er muß Auskunft darüber geben können, ob ein Feld des Schachbretts mit einer Figur besetzt ist oder nicht, und wenn ja, welche Figur (Bauer, Springer, Läufer, Turm, Dame oder König) von welcher Farbe auf diesem Feld steht.

   HINWEIS: Entwerfen Sie geeignete Teil-Datentypen, auf denen der Datentyp Schachbrett aufgebaut werden kann.

b) Implementieren Sie mit Hilfe des in Teil a) definierten Schachbrett-Datentyps je eine Modula-3 Prozedur, die für die Spielfigurarten Springer und Läufer ermittelt, welche Zugmöglichkeiten diese von einer bestimmten Position aus haben. Das heißt, auf welche freien Felder sie gezogen werden können und wo sie eine gegnerische Figur schlagen können.

   HINWEIS: Implementieren Sie geeignete Hilfsprozeduren (orientieren Sie sich an den Datentypen), die bei der Konstruktion der oben geforderten Prozeduren verwendet werden können.

c) Implementieren Sie ein Modula-3 Programm, in dem Sie eine Variable des in Teil a) entworfenen Schachbrett-Datentyps mit der oben dargestellten Stellung initialisieren und anschließend mit den in Teil b) implementierten Prozeduren für den schwarzen Springer auf Position (4, 5) und den weißen Läufer auf Position (3, 7) ermitteln, welche Zugmöglichkeiten diese haben. Das Programm soll diese Zugmöglichkeiten zusammen mit eventuell dabei zu schlagenden, gegnerischen Figuren ausgeben.

Programmierung **WS 1998/99** **Prof. H. Lichter**
M. Schnizler

Übung 9

Ausgabe:	Mo, 14.12.98	**Abgabe:**	Di, 22.12.98	**Globalübung:**	Di, 22.12.98
				Gruppen:	Do, 07.01.99
					Fr, 08.01.99

Aufgabe 9.1: Scanner *(10 Punkte)*

Teil eines jeden Übersetzers (Compilers) ist der "Scanner". Die Aufgabe eines Scanners ist es, eine Folge von Eingabezeichen in eine Folge von Texteinheiten, die sogenannten Symbole der Sprache, umzuwandeln. Im folgenden soll ein einfacher Scanner entwickelt werden.

Es gelten folgende Erkennungsregeln:

1. Die Menge der Sprach-Symbole besteht aus den Elementen `Bezeichner`, `Zahl`, `KleinerGleich`, `GroesserGleich`, `ErgibtSich`, weiteren Symbolen für die Einzelzeichen: `+`, `-`, `*`, `/`, `<`, `>`, `=`, `#`, `:`, `;`, sowie dem speziellen Symbol `Unbekannt` für unbekannte Zeichen.

2. Das Symbol `Bezeichner` wird beim Lesen einer mit einem Buchstaben beginnenden Folge von Buchstaben und Ziffern generiert.

3. Das Symbol `Zahl` wird beim Lesen einer Folge von Ziffern generiert.

4. Die Symbole `KleinerGleich`, `GroesserGleich` und `Wird` werden beim Lesen der entsprechenden Zeichenkombinationen (`<=`, `>=`, `:=`) generiert.

5. Beim Lesen eines der unter Punkt 1 genannten Einzelzeichen, wird ebenfalls ein entsprechendes Symbol generiert.

6. Leerzeichen, Tabulatoren sowie Zeilenenden werden übersprungen.

Implementieren Sie in Modula-3 mit Hilfe von Mengen einen Scanner für die angegebenen Erkennungsregeln. Dieser liest durch den Benutzer eingegebene Zeilen zeichenweise, erkennt dabei die entsprechenden Symbole und gibt diese nach jeder Zeile am Bildschirm aus. Definieren Sie dazu einen Datentyp Symbol, der die Art des Symbols beschreibt. Dieser speichert zusätzlich im Falle eines Bezeichners den Bezeichnernamen bzw. einer Zahl den Wert der Zahl. Das Programm soll abbrechen, sobald ein unbekanntes Zeichen gelesen wird; das heißt, wenn das Symbol `Unbekannt` generiert wird.

HINWEISE: Die Zeichen Tabulator, Zeilenvorschub (LF) und Wagenrücklauf (CR) lassen sich in Modula-3 mit den Sequenzen `"\t"`, `"\n"` und `"\r"` ausdrücken. Zeilenenden werden unter Unix (nur ein Zeichen: LF) und Windows (zwei Zeichen: CR/LF) unterschiedlich dargestellt. Die Verwendung eventuell vordefinierter Scanner-Operationen aus der Modula-3 Bibliothek ist **nicht zulässig**.

Aufgabe 9.2: Tennisrangliste *(10 Punkte)*

Ein Tennisverein möchte die Rangliste seiner Spieler mittels EDV verwalten. Da immer wieder neue Mitglieder aufgenommen werden, ist die Länge der Rangliste nicht von vornherein bekannt. Von jedem Spieler wird der Name sowie seine Mitgliedsnummer gespeichert. Implementieren Sie die Rangliste in der Programmiersprache Modula-3 als **einfach** verkettete Liste (vom besten zum schlechtesten Spieler). Dabei sollen folgende Operationen durch geeignete interaktive Prozeduren unterstützt werden:

1. Aufnehmen eines neuen Spielers (d.h. an das Ende der Rangliste setzen)
2. Streichen eines Spielers aus der Rangliste
3. Aktualisieren der Rangliste nach einem Spiel (Regel: Der Spieler A wird in der Rangliste vor dem Spieler B plaziert, wenn A gegen B gewonnen hat.)
4. Ausgeben der aktuellen Rangliste auf dem Bildschirm

Schreiben Sie ein geeignetes Modula-3 Rahmenprogramm und überprüfen Sie damit anhand einer Reihe von Eingaben Ihre Prozeduren.

Programmierung **WS 1998/99** **Prof. H. Lichter**
M. Schnizler

Übung 10

Ausgabe: Mo, 21.12.98 **Abgabe:** Di, 12.01.99 **Globalübung:** Di, 12.01.99
Gruppen: Do, 14.01.99
Fr, 15.01.99

Aufgabe 10.1: Ruprechts Lagerverwaltung *(10 Punkte)*

Der Weihnachtsmann und sein Knecht Ruprecht wollen ihr Geschenkelager effizienter verwalten. Um die beiden in ihrem Vorhaben zu unterstützen, sollen Sie eine Lagerverwaltung in Modula-3 implementieren. Ruprechts Lager enthält unterschiedliche Arten von Geschenken (zum Beispiel Ball, Schaukelpferd, Lokomotive, Waggon, Puppe). Zu jeder Geschenkart sind normalerweise mehrere verschiedene Geschenke im Lager vorhanden, die sich beispielsweise durch die Farbe (zum Beispiel grüne, rote und blaue Bälle) oder die Größe (zum Beispiel kleine, mittlere und große Puppen) unterscheiden.

Entwerfen Sie die notwendigen Datentypen für die Lagerverwaltung. Dabei muß es sowohl möglich sein, beliebig viele neue Arten von Geschenken in das Lager aufzunehmen als auch zu jeder Art von Geschenk beliebig viele Geschenke ein- und auszulagern.

Implementieren Sie für die entworfenen Datentypen die folgenden Operationen:

- Aufnehmen einer neuen Geschenkart
- Einlagern eines Geschenks einer vorhandenen Geschenkart
- Entnehmen eines Geschenks aus dem Lager (wenn dadurch weniger als fünf Geschenke dieser Art im Lager verbleiben, muß eine entsprechende Meldung ausgegeben werden)
- Ausdrucken des kompletten Lagerbestands (jedes Geschenk mit Beschreibung für alle Geschenkarten)

Testen Sie die von Ihnen entworfenen Operationen mit einem geeigneten Hauptprogramm.

HINWEISE: Verwenden Sie als Datentyp für die Geschenkart und die Beschreibung eines Geschenks den Typ `TEXT`. Verwenden Sie für Vergleiche zwischen diesen Werten die Operation `Compare` aus dem Modul `Text`.

Aufgabe 10.2: Zählen von Wörtern *(10 Punkte)*

Implementieren Sie ein Modula-3 Programm, das alle unterschiedlichen Wörter, die in einer Textdatei vorkommen, ermittelt und zählt, wie oft jedes Wort in der Datei vorkommt. Die Liste der erkannten Wörter mit der Häufigkeit des Vorkommens in der Datei soll lexikographisch sortiert in einer Ergebnisdatei ausgegeben werden. Das Programm fragt den Benutzer zu Beginn nach den Namen der auszuwertenden Textdatei und der Ergebnisdatei. Anschließend wird die Textdatei mit dem angegebenen Namen geöffnet und eine Liste der enthaltenen Wörter und ihrer jeweiligen Anzahl durch sequentielles Lesen der Datei aufgebaut. Die in der Liste gesammelten Daten werden am Schluß in die vom Benutzer angegebene Ergebnisdatei geschrieben.

Testen Sie ihr Programm mit der im Netz bzw. CIP-Pool zusammen mit dem Aufgabenblatt 10 bereitgestellten Textdatei "Ribbeck.txt" und geben Sie die ausgedruckte Ergebnisdatei zusammen mit Ihrem Programm ab.

HINWEISE: Verwenden Sie für die Arbeit mit Dateien die in der Vorlesung vorgestellten Operationen aus den Modulen `IO`, `Rd` und `Wr`.

Um einzelne Wörter der Textdatei zu erkennen, verwenden Sie bitte den Scanner aus Aufg. 9.1, den Sie für das Lesen aus einer Datei (weiteres Symbol `EOF`) anpassen müssen. Da im Kontext dieser Aufgabe nur die Symbole `Bezeichner`, `Unbekannt` und `EOF` notwendig sind, können Sie den Scanner auf die Erkennung dieser Symbole reduzieren.

Verwenden Sie für Vergleiche zwischen einzelnen Wörtern bei der lexikographischen Sortierung die Operation `Compare` aus dem Modul `Text`.

Programmierung **WS 1998/99** **Prof. H. Lichter**
M. Schnizler

Übung 11

Ausgabe:	Mo, 11.01.99	**Abgabe:**	Di, 19.01.99	**Globalübung:**	Di, 19.01.99
				Gruppen:	Do, 21.01.99
					Fr, 22.01.99

Aufgabe 11.1: Matrix mit Prozedurtypen *(5 Punkte)*

Implementieren Sie eine alternative Lösung zu Aufg. 8.1, bei der die Art der Matrix-Transformation (Spiegelung an vertikaler Symmetrieachse, Drehung im Uhrzeigersinn, ...), mit Hilfe eines Prozedurtyp-Parameters angegeben wird. Das heißt, abhängig davon, wie die Matrix umgeformt werden soll, muß jeweils eine andere Funktion als aktueller Parameter an die umformende Prozedur übergeben werden.

Aufgabe 11.2: Tennisrangliste als Objektmodul *(15 Punkte)*

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

1. Aufnehmen eines neuen Spielers (d.h. an das Ende der Rangliste setzen)
2. Streichen eines Spielers aus der Rangliste
3. Aktualisieren der Rangliste nach einem Spiel (Regel: Der Spieler A wird in der Rangliste vor dem Spieler B plaziert, wenn A gegen B gewonnen hat.)
4. Ausgeben der aktuellen Rangliste auf dem Bildschirm

Ermitteln Sie außerdem Testfunktionen, die in der Schnittstelle enthalten sein müssen, um mit dem Tennisranglisten-Objektmodul zu arbeiten. Überlegen Sie sich dazu insbesondere die Vorbedingungen, die jeweils erfüllt sein müssen, um die obigen Operationen zu verwenden.

Realisieren Sie für diese Schnittstelle zwei verschiedene Implementierungen:

- Implementierung 1: Passen Sie die für Aufg. 9.2 entwickelte, auf einer einfach-verketteten Liste basierende Lösung an das Objektmodul an.
- Implementierung 2: Verwenden Sie für die Datenstruktur der Tennisrangliste ein Feld mit fester Größe.

Schreiben Sie ein geeignetes Modula-3 Rahmenprogramm und überprüfen Sie damit anhand einer Reihe von Eingaben (Grenzfälle berücksichtigen!) Ihre beiden Implementierungen.

HINWEIS: Trennen Sie die Daten eines Spielers von der Datenstruktur für die Rangliste, indem sie zusätzlich einen ADT-Modul für einen Datentyp Spieler definieren.

Programmierung **WS 1998/99** **Prof. H. Lichter**
M. Schnizler

Übung 12

Ausgabe:	Mo, 18.01.99	**Abgabe:**	Di, 26.01.99	**Globalübung:**	Di, 26.01.99
				Gruppen:	Do, 28.01.99
					Fr, 29.01.99

Aufgabe 12.1: Datenbank für Personendaten *(10 Punkte)*

Implementieren Sie in Modula-3 eine einfache Datenbank für Personendaten, bestehend aus Name, Vorname und Alter der Person. Entwerfen Sie zunächst eine Datenstruktur, die geeignet ist, eine beliebige Menge solcher Personendaten zu erfassen. Implementieren Sie zu dieser Datenstruktur eine Prozedur, die es erlaubt, anhand eines Auswahlkriteriums Teilmengen dieser Datenbank zu bestimmen, welche anschließend ausgegeben werden können. Das Auswahlkriterium soll dabei als Parameter eines geeigneten Prozedurtyps an die Prozedur übergeben werden.

Implementieren Sie insbesondere folgenden Anfragen:

- Alle Personen, die jünger als 18 Jahre alt sind.
- Alle Personen mit Vorname "Bert" und Alter 20 Jahre.
- Alle Personen, deren Name mit "A" anfängt.

Testen Sie die von Ihnen entworfene Datenstruktur und Prozeduren mit Hilfe eines geeigneten Modula-3 Hauptprogramms.

HINWEIS: Implementieren Sie für die Datenstruktur der Personendaten nur die für die Aufgabe notwendigen Operationen. Sie brauchen beispielsweise keine Operation für das Löschen eines Elements zu definieren.

Aufgabe 12.2: ADT für Geldbeträge *(10 Punkte)*

Implementieren Sie in Modula-3 einen abstrakten Datentyp (ADT) für Geldbeträge. Der Datentyp muß geeignet sein, um Beträge in unterschiedlichen Landeswährungen zu speichern und Berechnungen mit solchen Beträgen auszuführen.

Der ADT muß folgende Operationen anbieten:

- Erzeugen von Geldbeträgen in den Währungen CHF, DM, FF, EUR, GBP und USD.
- Umtauschen eines Geldbetrags einer Währung in eine andere Währung.
- Addieren bzw. subtrahieren zweier Geldbeträge in verschiedenen Währungen.
- Multiplizieren eines Geldbetrags mit einer REAL-Zahl (z.B. für Rabatt-Berechnung).
- Ausgeben des Geldbetrags mit der zugrundeliegenden Währung als Text.
- Vergleiche zwischen zwei Geldbeträgen (=, <, >).

Testen Sie Ihren ADT mit Hilfe eines geeigneten Modula-3 Hauptprogramms.

Programmierung **WS 1998/99** **Prof. H. Lichter**
M. Schnizler

Übung 13

Ausgabe:	Mo, 25.01.99	**Abgabe:**	Di, 02.02.99	**Globalübung:**	Di, 02.02.99
				Gruppen:	Do, 04.02.99
					Fr, 05.02.99

Aufgabe 13.1: Einmaleins-Taschenrechner *(10 Punkte)*

Implementieren Sie in Modula-3 ein Modul, das die Operationen eines einfachen Taschenrechners, mit dem beispielsweise Kinder das Einmaleins üben können, bereitstellt. Der Taschenrechner beherrscht den Zahlenbereich der ganzen Zahlen von 0 bis 100 und erlaubt die Addition, Subtraktion, Multiplikation sowie die ganzzahlige Division (DIV) mit Zahlen aus diesem Bereich.

Überlegen Sie sich die Ausnahmesituationen, die bei der Ausführung dieser Operationen auftreten können und definieren Sie hierfür Ausnahmen in Ihrem Taschenrechner-Modul. Implementieren Sie schließlich ein Modula-3 Hauptprogramm, das einem Benutzer erlaubt, mit dem Taschenrechner zu arbeiten. Achten Sie hierbei auf die korrekte Behandlung der dabei möglicherweise auftretenden Ausnahmen.

Aufgabe 13.2: Getränkeautomat *(10 Punkte)*

Implementieren Sie in Modula-3 ein Objektmodul für die Simulation eines Getränkeautomaten:

> Der Automat enthält nur eine Getränkesorte. Außerdem akzeptiert er nur Ein-Euro-Münzen und ein Getränk kostet genau einen Euro. Es kann höchstens eine Münze eingeworfen werden, bevor ein Getränk entnommen wird. Desweiteren enthält der Automat nur eine begrenzte Anzahl von Getränken, die einzeln nachgefüllt werden können.

Implementieren Sie das Objektmodul und überlegen Sie sich sinnvolle Vor- und Nachbedingungen, die für die einzelnen Operationen des Automaten gelten müssen. Überlegen Sie auch, ob es Invarianten gibt, die immer gelten müssen. Führen Sie für diese Zwecke entsprechende Testfunktionen in Ihrem Objektmodul ein.

Verwenden Sie für Ihre Implementierung die Operationen des in der Vorlesung vorgestellten Moduls `Assertion`, um an den entsprechenden Stellen Ihrer Implementierung die Vor- und Nachbedingungen sowie Invarianten zu überprüfen. Testen Sie das von Ihnen entwickelte Objektmodul mit Hilfe eines geeigneten Hauptprogramms, in dem Sie verschiedene gültige und ungültige Aufrufsequenzen an das Objektmodul richten.

HINWEIS: Eine kommentierte Version des Moduls `Assertion` finden Sie zusammen mit diesem Übungsblatt auf den WWW-Seiten zur Vorlesung Programmierung.