LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **1. Übungsblatt**

Musterlösung

Aufgabe 1.1

```
Nimm das Telefonbuch
Wenn die gesuchte Nummer im Ausland ist:
     Schlage die Nummer der Auslandsauskunft nach.
     Nimm den Telefonhörer ab.
     Wähle die Nummer der Auslandsauskunft.
sonst
     Schlage die Nummer der Inlandsauskunft nach.
     Nimm den Telefonhörer ab.
     Wähle den Nummer der Inlandsauskunft.
Wenn das Besetztzeichen ertönt:
     Solange bis nicht mehr besetzt ist:
          Lege den Hörer auf.
          Nimm den Telefonhörer ab.
          Wähle die zuletzt gewählte Nummer erneut.
Wenn Du in die Warteschlange kommst:
     Solange wie Du in der Warteschlange bist:
          Warte.
Solange bis sich jemand meldet:
     Warte.
Nenne den Namen des gewünschten Teilnehmers.
Nenne die Adresse des gewünschten Teilnehmers.
Wenn die Nummer der Auskunft bekannt ist:
     Solange bis die Nummer angesagt wird:
          Warte:
     Notiere die Nummer, die angesagt wird.
     Überprüfe die Nummer, die angesagt wird.
Lege den Telefonhörer auf.
```

Aufgabe 1.2

Da der Rechner keine Befehle zur Multiplikation oder zum Potenzieren kennt, muß man das Quadrieren einer gegebenen Zahl N auf das N-fache Addieren zurückführen:

1	`LoadAdr`	`1, 20`	Lade den Wert N, der quadriert werden soll aus der Speicherstelle 20 in das Register 1
3	`LoadVal`	`2, 0`	Lade den Wert 0 in das Register 2. Hier steht später das Ergebnis
3	`LoadVal`	`3, 0`	Lade den Wert 0 in das Register 3. Hier zählt man mit, wie oft man den Wert aus Register 1 schon zu Register 3 hinzuaddiert hat.
4	`Compare`	`1, 3`	Vergleiche die Werte in Register 1 und 3.
5	`JmpEqual`	`9`	Falls sie gleich sind, hat man den Wert N bereits N-mal aufaddiert und Register 2 enthält damit $N*N = N^2$. In diesem Fall gehe zu Speicherstelle 9.
6	`Add`	`2, 1`	Addiere den Wert aus Reg. 1 zu Reg. 2.
7	`Inc`	`3`	Erhöhe Register 3 um eins.
8	`Jmp`	`4`	Gehe wieder zu Schritt 4.
9	`StoreAdr`	`2, 21`	Schreibe das Ergebnis aus Reg. 2 an Speicherstelle 21.

Aufgabe 1.3 folgt aus Platzgründen nach Aufgabe 1.4.

Aufgabe 1.4

Umrechnung von 72329 und 18,453125 in das Binärsystem:

2^{16}	65536	72329 - **1** · 2^{16}	=	6793		
2^{15}	32768	6793 - **0** · 2^{15}	=	6793		
2^{14}	16384	6793 - **0** · 2^{14}	=	6793		
2^{13}	8192	6793 - **0** · 2^{13}	=	6793		
2^{12}	4096	6793 - **1** · 2^{12}	=	2697		
2^{11}	2048	2697 - **1** · 2^{11}	=	649		
2^{10}	1024	649 - **0** · 2^{10}	=	649		
2^{9}	512	649 - **1** · 2^{9}	=	137		
2^{8}	256	137 - **0** · 2^{8}	=	9		
2^{7}	128	9 - **1** · 2^{7}	=	9		
2^{6}	64	9 - **0** · 2^{6}	=	9		
2^{5}	32	9 - **0** · 2^{5}	=	9		
2^{4}	16	9 - **0** · 2^{4}	=	1	18,453125 - **1** · 2^{4}	= 2,453125
2^{3}	8	1 - **1** · 2^{3}	=	1	2,453125 - **0** · 2^{3}	= 2,453125
2^{2}	4	1 - **0** · 2^{2}	=	1	2,453125 - **0** · 2^{2}	= 2,453125
2^{1}	2	1 - **0** · 2^{1}	=	1	2,453125 - **1** · 2^{1}	= 0,453125
2^{0}	1	1 - **1** · 2^{0}	=	0	0,453125 - **0** · 2^{0}	= 0,453125
2^{-1}	0,5				0,453125 - **0** · 2^{-1}	= 0,453125
2^{-2}	0,25				0,453125 - **1** · 2^{-2}	= 0,203125
2^{-3}	0,125				0,203125 - **1** · 2^{-3}	= 0,078125
2^{-4}	0,0625				0,078125 - **1** · 2^{-4}	= 0,015625
2^{-5}	0,03125				0,015625 - **0** · 2^{-5}	= 0,015625
2^{-6}	0,015625				0,015625 - **1** · 2^{-6}	= 0

Ergebnis: **10001101010001001**

Ergebnis: **10010,011101**

Umrechnung von 129700 und 325,060546875 in das Oktalsystem:

8^5	32768	129700	-	**3**	·	8^5	=	31396						
8^4	4096	31396	-	**7**	·	8^4	=	2724						
8^3	512	2724	-	**5**	·	8^3	=	164						
8^2	64	164	-	**2**	·	8^2	=	36	325,060546875	-	**5**	· 8^2	=	5,060546875
8^1	8	36	-	**4**	·	8^1	=	4	5,060546875	-	**0**	· 8^1	=	5,060546875
8^0	1	4	-	**4**	·	8^0	=	0	5,060546875	-	**5**	· 8^0	=	0,060546875
8^{-1}	0,125								0,060546875	-	**0**	· 8^{-1}	=	0,060546875
8^{-2}	0,015625								0,060546875	-	**3**	· 8^{-2}	=	0,013671875
8^{-3}	0,001953125								0,013671875	-	**7**	· 8^{-3}	=	0
		Ergebnis: **375244**							Ergebnis: **505,037**					

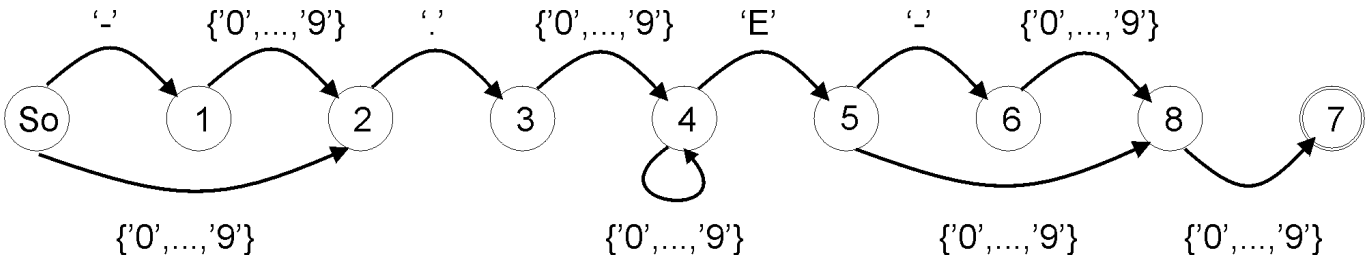
Umrechnung von 2729309 und 31300424,05078125 in das Hexadezimalsystem:

Das Hexadezimalsystem enthält neben den Ziffern 0 bis 9 die Buchstaben A bis F als zusätzliche Ziffern, um die Werte 10 bis 15 einer Stelle ausdrücken zu können.

16^5	1048576	2729309	-	**2**	·	16^5	=	632157
16^4	65536	632157	-	**9**	·	16^4	=	42333
16^3	4096	42333	-	**10**	·	16^3	=	1373
16^2	256	1373	-	**5**	·	16^2	=	93
16^1	16	93	-	**5**	·	16^1	=	13
16^0	1	13	-	**13**	·	16^0	=	
		Ergebnis: **29A5513**						

16^6	16777216	31300424,05078125	-	**1**	·	16^6	=	14523208,05078125
16^5	1048576	14523208,05078125	-	**13**	·	16^5	=	891720,05078125
16^4	65536	891720,05078125	-	**13**	·	16^4	=	39752,05078125
16^3	4096	39752,05078125	-	**9**	·	16^3	=	2888,05078125
16^2	256	2888,05078125	-	**11**	·	16^2	=	72,05078125
16^1	16	72,05078125	-	**4**	·	16^1	=	8,05078125
16^0	1	8,05078125	-	**8**	·	16^0	=	0,05078125
16^{-1}	0,0625	0,05078125	-	**0**	·	16^{-1}	=	0,05078125
16^{-2}	0,00390625	0,05078125	-	**13**	·	16^{-2}	=	0
		Ergebnis: **1DD9B48,0D**						

Aufgabe 1.3



LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

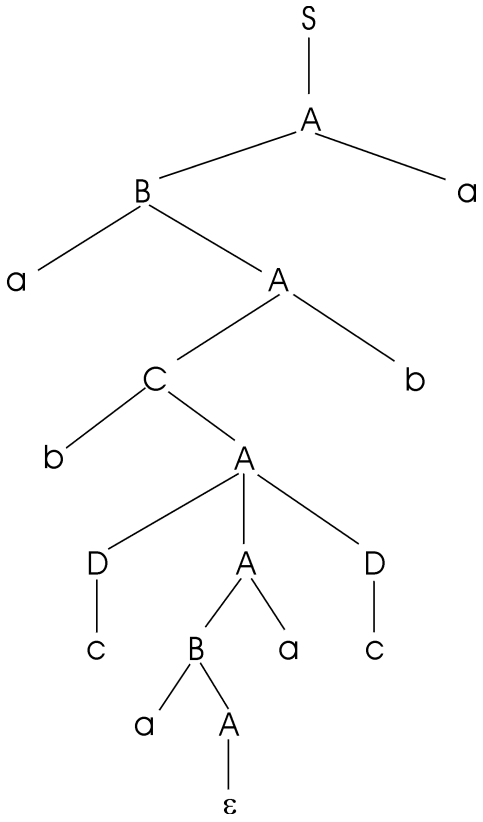
Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **2. Übungsblatt**

Musterlösung

Aufgabe 2.1

a) Die Grammatik definiert die Sprache alle Palindrome über dem Alphabet {a,b,c}. Als Palindrome werden Wörter bezeichnet, die beim Lesen von hinten nach vorne die selbe Folge von Buchstaben aufweisen wie beim Lesen von vorne nach hinten.
Des Ableitungsbaum für das Beispielwort "abcaacba" sieht wie folgt aus:



b) G = ({A},{a,b},P,S)

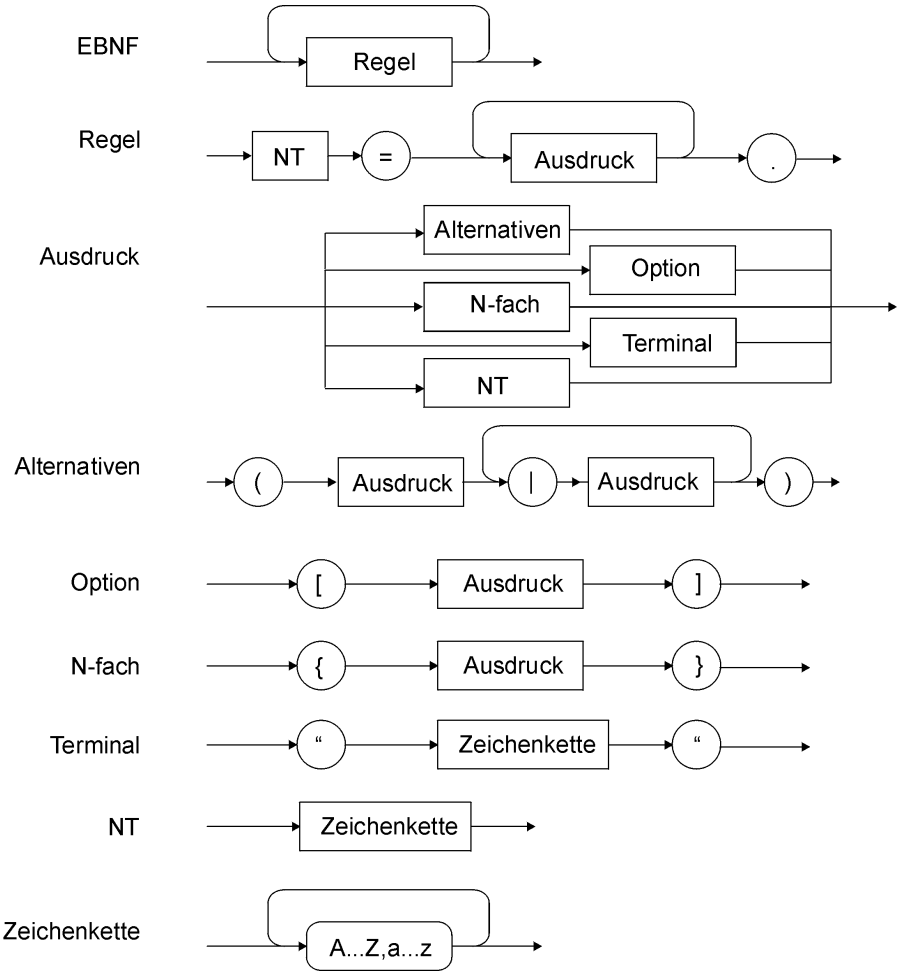
P = { S → A,
A → aAb,
A → ab }

c) G = ({A,B},{0, ... , 9, +, – },P , S)

P = { S → ABA,
A → ABA,
A → 0 , A → 1 ,
A → 2 , A → 3 ,
A → 4 , A → 5 ,
A → 6 , A → 7 ,
A → 8 , A → 9 ,
B → –, B → + }

Aufgabe 2.2

a) Syntaxdiagramme für EBNF:



b) EBNF für EBNF:

EBNF = **Regel** { **Regel** } .
Regel = **NT** "=" **Ausdruck** { **Ausdruck** } " **.** " .
Ausdruck = (**Alternative** | **Option** | **N-fach** | **NT** | **Terminal**) .
Alternative = "(" **Ausdruck** "|" **Ausdruck** { "|" **Ausdruck** } ")" .
Option = "[" **Ausdruck** "]" .
N-fach = "{" **Ausdruck** "}" .
Terminal = "'" **Zeichenkette** "'" .
NT = **Zeichenkette** .
Zeichenkette = ("a" | ... | "z" | "A" | ... | "Z") { ("a" | ... | "z" | "A" | ... | "Z") } .

Aufgabe 2.3

Die Addition von zwei Zahlen, die in Strichdarstellung dargestellt und durch ein Leerzeichen getrennt sind, wird durchgeführt, indem man die zweite Zahl um eine Position nach links auf dem Band verschiebt. Dazu überschreibt man das Leerzeichen mit einem Strich und rückt die rechte Endmarkierung um ein Feld nach links.

Die unten angegebene Tabelle funktioniert auch für den Fall, daß beide Zahlen gleich null sind. Auf dem Band stehen in diesem Fall hintereinander die Zeichen #B#. Allerdings kann dann die Aufgabenstellung nich mehr vollständig erfüllt werden, da das Ergebnis der Addition ebenfalls null Striche besitzt und es somit keinen ersten Strich der Strichdarstellung gibt, auf dem sich die Maschine nach Ende der Berechnung befinden soll. Die Maschine steht dann auf der rechten Endmarke.

Im Zustand Z_3 wird davon ausgegangen, daß links vom der rechten Endmarke ein Strich auf dem Band steht. Das ist immer der Fall, denn entweder hat die s(b) mindestens einen Strich, oder es ist der Strich, den wir über das Blank geschrieben haben.

Zust.	**gelesen**	**neuer Zust.**	**schreibe**	**Bewegung**	**Kommentar**
Z_1	|	Z_1	|	R	Gehe bis zum Blank und überschreibe
Z_1	B	Z_2	|	R	das Blank mit einem Strich.
Z_2	|	Z_2	|	R	Laufe nach rechts bis zum Endmarke '#'
Z_2	#	Z_3	B	L	und überschreibe sie mit einem Blank.
Z_3	|	Z_4	#	L	Schreibe neue Endmarke 1 Feld links davon.
Z_4	|	Z_4	|	L	Laufe nach links bis zur linken Endmarke.
Z_4	#	Z_C	#	R	Von der linken Endmarke gehe eine Schritt nach rechts auf den ersten Strich der Strichdarstellung.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **3. Übungsblatt**

Musterlösung

Aufgabe 3.1

G = (N, T, P, S) mit

N = {**T***, **C***, **B***, **S***, **D***, **F***, **S**, **A‘**, **A‘‘**, **G‘**, **K‘**, **D‘**, **Z‘**, **T‘**, **M‘**, **J‘**, **W‘**}

T = {alle Zeichen auf der Tastatur, insbesondere \n = Zeilenumbruch, _ = Leerzeichen}

P = {

S	➔ **T***\n**C***\n**B***\n**S***\n**D***:\n**F***:	
T*	➔ TO:_**A‘‘**	
C*	➔ CC:_**A‘‘** | CC:	
B*	➔ BCC:_**A‘‘** | BCC:	
S*	➔ SUBJECT:_**W‘** | SUBJECT:	
W‘	➔ **K‘** | **G‘** | **Z‘** | **K’W‘** | **G’W‘** | **Z’W‘**	definiert normalen Text, der alle Zeichen von der Tastatur enthalten kann
D*	➔ **T‘**:**M’**:**J‘**	
T‘	➔ 01 | ... | 31	
M‘	➔ 01 | ... | 12	
J‘	➔ 00 | ... | 99	
F*	➔ FROM:_**A‘**	
A‘	➔ **G’K‘**_**G’K‘**_<**K‘@D‘**> | **K‘@D‘**	name in Kleinbuchstaben
A‘‘	➔ **A‘** | **A‘**, **A‘‘**	Adressenliste
D‘	➔ **K‘** | **K‘**.**D‘**	domain in Kleinbuchstaben
G‘	➔ A | ... | Z | A**G‘** | ... | Z**G‘**	Großbuchstabenfolgen
K‘	➔ a | ... | z | a**K‘** | ... | z**K‘**	Kleinbuchstabenfolgen
Z‘	➔ ! | ...	alle anderen Zeichen

}

Aufgabe 3.2

```
MODULE Reverse EXPORTS Main;

FROM Stdio IMPORT stdout,stdin;
FROM Wr IMPORT PutText;
FROM Rd IMPORT GetLine;
FROM Text IMPORT Equal;

(*-------------------------------------------------------------------------------
  Name:      InputUntilStop

  Aufgabe:   Die Funktion erhaelt als Eingabe einen Text im Parameter word.
             (Anmerkung: Wir definieren der Einfachheit halber, dass fuer
             uns ein Wort genau das ist, was wir in diesem Parameter
             uebergeben, und ueberpruefen nicht, ob es sich im sprachlichen
             Sinne tatsaechlich um ein einzelnes Wort handelt.)
             Wenn ein Wort ungleich "\STOP" uebergeben wird, ruft sich
             die Funktion selbst wieder auf. Den Parameter fuer diesen
             Aufruf, das naechste Wort, erhaelt die Funktion durch den
             Aufruf der Funktion GetLine.
             Diese Kette von rekursiven Aufrufen terminiert genau dann,
             wenn der Benutzer irgendwann einmal "\STOP" eingibt. Jedesmal
             wenn eine Rekursionsebene beendet ist, wird als naechster
             Befehl in der Ebene, in der der Aufruf erfolgte, PutText
             aufgerufen und das Wort, das dieser Ebene als Parameter
             uebergeben wurde, wird ausgegeben. Da bei der Abfolge der
             Rueckspruenge die Rekursionsebenen in umgekehrter
             Reihenfolge des Aufrufs durchlaufen werden, kommt die
             Umkehrung der Worte bei der Ausgabe zustande.

  Parameter: word - das vom Benutzer eingegebene Wort

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE InputAndReverse(word: TEXT) =
  BEGIN
    IF NOT Equal (word,"\\STOP") THEN
      InputAndReverse(GetLine(stdin));   (* Aufruf der naechsten Rekursion *)
      PutText(stdout,word & " ")         (* Gib das Wort aus, das als Parameter
                                            uebergeben wurde. *)
    END;
END InputAndReverse;
(*-------------------------------------------------------------------------------
  Hauptprogramm

  Das Hauptprogramm enthaelt den initialen Aufruf der Funktion InputAndReverse
-------------------------------------------------------------------------------*)
BEGIN
  InputAndReverse(GetLine(stdin));
END Reverse.
(*-------------------------------------------------------------------------------
 PROGRAMMENDE
-------------------------------------------------------------------------------*)
```

Aufgabe 3.3

a) Programmtext

```
MODULE Pascal EXPORTS Main;

FROM Stdio IMPORT stdout,stdin;
FROM Wr IMPORT PutText,Flush;
FROM Rd IMPORT GetLine;
IMPORT Fmt;
IMPORT Scan;
(*-----------------------------------------------------------------------------
  Name:      Coefficient

  Aufgabe:   Die Funktion berechnet den Koeffizienten an der Position n,i
             im Pascalschen Dreieck. Es wird zwischen den einfachen Faellen
             unterschieden, in denen der Koeffizient unmittelbar angegeben
             werden kann und den komplexeren Faellen, in denen er aus
             den Koeffizienten der Zeile darueber berechnet werden muss.
             In den einfachen Faelle wird der entsprechende Wert direkt
             zurueckgegeben. In den komplexeren Faellen, ruft sich die Funktion
             zunaechst selbst wieder auf, um die beiden Koeffizienten der
             darueberliegenden Zeile des Dreiecks zu berechnen, und addiert
             sie dann.

  Parameter: n - die Zeile im Pascalschen Dreieck
             i - die Position des Koeffizienten in der Zeile n

  Rueckgabe: der Koeffizient fuer die uebergebenen Werte n und i
-----------------------------------------------------------------------------*)
PROCEDURE Coefficient(n,i:CARDINAL):CARDINAL =
  BEGIN
    IF i>n THEN               (* Fuer n>i haben alle Koeffizienten den Wert 0 *)
      RETURN 0;
    ELSIF i=0 OR i=n THEN     (* Fuer i=0 oder i=n haben sie den Wert 1 *)
      RETURN 1;
                              (* In allen anderen Faellen berechnet sich der
                                 Wert durch Addition der beiden im Dreieck
                                 darueber liegenden Werte. *)
    ELSE
      RETURN Coefficient(n-1,i) + Coefficient(n-1, i-1);
    END;
  END Coefficient;
(*-----------------------------------------------------------------------------
  Hauptprogramm
-----------------------------------------------------------------------------*)
BEGIN
  PutText(stdout,"Bitte geben Sie die Werte fuer n und i ein: \n");
  Flush(stdout);
  PutText(stdout,"Der Koeffizient hat den Wert:" &
                 Fmt.Int( Coefficient(Scan.Int(GetLine(stdin)),Scan.Int(GetLine(stdin)))));
  PutText(stdout,"\n");
END Pascal.
```

b) Abfolge der Funktionsaufrufe

```
Coefficient (5,2)
    Coefficient (4,2)
        Coefficient (3,2)
            Coefficient (2,2)
            RETURN 1
            Coefficient (2,1)
                Coefficient (1,1)
                RETURN 1
                Coefficient(1,0)
                RETURN 1
            RETURN 2
        RETURN 3
        Coefficient(3,1)
            Coefficient (2,1)
                Coefficient (1,1)
                RETURN 1
                Coefficient(1,0)
                RETURN 1
            RETURN 2
        RETURN 3
    RETURN 6
    Coefficient (4,1)
        Coefficient (3,1)
            Coefficient (2,1)
                Coefficient (1,1)
                RETURN 1
                Coefficient(1,0)
                RETURN 1
            RETURN 2
        RETURN 3
        Coefficient (3,0)
        RETURN 1
    RETURN 4
RETURN 10
```

Aufgabe 3.4

```
MODULE Replace EXPORTS Main;

FROM Stdio IMPORT stdout,stdin;
FROM Wr IMPORT PutText;
FROM Rd IMPORT GetLine;
FROM Text IMPORT Length, GetChar, Sub, Equal, FromChar, Empty;

(*---------------------------------------------------------------------------
  Name:       InputUntilStop

  Aufgabe:    Die Funktion dient zum zeilenweisen Einlesen der Eingaben des
              Benutzers. Sie ist rekursiv definiert und besitzt zwei Parameter
              Im Parameter lines wird der bisher eingegebene Text uebergeben.
              Im Parameter newLine wird der in der vorherigen Rekursionsebene
              eingegebene Text uebergeben. Die Funktione testet als erstes, ob
              dieser im Parameter newLine uebergebene Text das Wort "\STOP" ist.
              Dies ist die Abbruchbedingung der Rekursion. In diesem Fall wird
              keine weitere Eingabe gelesen. An den Text, der im Parameter
              lines uebergeben wurden, wird ein Newline-Zeichen angehaengt.
              Beides zusammen wird als Ergebnis der Funktion zurueckgegeben.
              Falls etwas anderes als "\STOP" eingegeben wurde, ruft sich
              die Funktion selbst wieder auf. Als erster aktueller Parameter
              wird bei diesem Aufruf ein String uebergeben, der sich aus
              dem Verketten des Parameters lines, einem Newline Zeichen und
              dem Parameter newline ergibt. Der zweite Parameter ist die
              naechste vom Benutzer einzugebene Zeile, die ueber den
              Aufruf der Funktion GetLine gelesen wird. Das Ergebnis dieses
              Aufrufs wird dann als Ergebnis der Funktion zurueckgegeben.

  Parameter:  lines      - die Zeilen, die bereits gelesen wurden
              newLine    - die zuletzt vom Benutzer eingegebene Zeile

  Rueckgabe:  ein Texte, der die vom Benutzer eingegeben Zeilen, getrennt
              jeweils durch ein Newline Zeichen, enthaelt
---------------------------------------------------------------------------*)

PROCEDURE InputUntilStop (lines: TEXT; newLine: TEXT):TEXT =
  BEGIN
    IF Equal (newLine,"\\STOP") THEN
      RETURN lines & "\n";
    ELSE
      RETURN InputUntilStop (lines & "\n" & newLine, GetLine(stdin));
    END;
END InputUntilStop;

(*---------------------------------------------------------------------------
  Name:       ReplaceVowels

  Aufgabe:    Die Funktion erhaelt als Eingabe einen Text im Parameter
              remainingText. Wenn der Text leer ist, gibt sie ebenfalls einen
              leeren Text zurueck. Ansonsten liest sie den ersten Buchstaben
              des Textes. Wenn es ein Vokal ist, gibt sie die Verkettung
              des Textes "_" mit dem Text zuruck, den die Funktion beim
              rekursiven Aufruf fuer den Rest des Textes, also fuer das
              zweite bis n-te Zeichen liefert. Wenn das erste Zeichen kein
              Vokal ist, verkettet die Funktion dieses Zeichen unveraendert mit
              dem Resultat des rekursiven Aufrufs auf dem Rest des Textes.
              Die Funktion schaut sich also in einem gegebenen Text das erste
              Zeichen an, ersetzt es, wenn es ein Vokal ist, und ruft sich dann
              selbst wieder auf, um die selbe Ersetzung fuer das erste Zeichen
              des restlichen Textes durchzufuehren usw.

  Parameter:  remainingText - der zu bearbeitende Text

  Rueckgabe:  der Inhalt von remainingText, wobei alle Vokale durch
              Unterstriche '_' ersetzt wurden.
---------------------------------------------------------------------------*)

PROCEDURE ReplaceVowels (remainingText: TEXT):TEXT =
  BEGIN
    IF Empty(remainingText) THEN
      RETURN "";
    ELSE
      CASE GetChar (remainingText,0) OF
      | 'e','E','o','O','a','A','i','I','u','U' =>
        RETURN "_" & ReplaceVowels(Sub(remainingText,1,Length(remainingText)-1));
      ELSE
        RETURN FromChar(GetChar(remainingText,0)) &
               ReplaceVowels(Sub(remainingText,1,Length(remainingText)-1));
      END
    END;
END ReplaceVowels;
```

```
(*-------------------------------------------------------------------------------
  Hauptprogramm

  Hier wird die Funktion PutText verwendet, um das Ergebnis der Funktion
  ReplaceVowels auszugeben, die wiederum auf dem Ergebnis der Funktion
  InputUntilStop arbeitet. InputUntilStop wird mit zwei leeren Strings
  als aktuellen Parametern aufgerufen, da es zu diesem Zeitpunkt noch
  keine bisher eingegebene Zeilen (erster Parameter) und keine neue Zeile
  (zweiter Parameter) gibt.
-------------------------------------------------------------------------------*)

BEGIN
  PutText(stdout, ReplaceVowels(InputUntilStop("","")));
END Replace.
```

Eingabe des Benutzers:

Schreiben Sie ein rein funktionales rekursives Programm, dass die Worte eines Textes einzeln vom Benutzer einliest und anschliessend in umgekehrter Reihenfolge wieder ausgibt. Die Reihenfolge der Buchstaben innerhalb der Worte soll erhalten bleiben. Das Programm soll solange weitere Worte einlesen, bis der Benutzer die Zeichenfolge "\STOP" eingibt. Danach sollen die Worte hintereinander ohne Zeilenumbruch und jeweils durch ein Leerzeichen getrennt ausgegeben werden. Das abschliessende Wort "\STOP" soll nicht ausgegeben werden.
\STOP

Ausgabe des Programms:

Schr__b_n S__ __n r__n f_nkt__n_l_s r_k_rs_v_s Pr_gr_mm, d_ss d__ W_rt_ __n_s T_xt_s __nz_ln v_m B_n_tz_r __nl__st _nd _nschl__ss_nd _n _mg_k_hrt_r R__h_nf_lg_ w__d_r __sg_bt. D__ R__h_nf_lg_ d_r B_chst_b_n _nn_rh_lb d_r W_rt_ s_ll _rh_lt_n bl__b_n. D_s Pr_gr_mm s_ll s_l_ng_ w__t_r_ W_rt_ __nl_s_n, b_s d_r B_n_tz_r d__ Z__ch_nf_lg_ "\ST_P" __ng_bt. D_n_ch s_ll_n d__ W_rt_ h_nt_r__n_nd_r _hn_ Z__l_n_mbr_ch _nd j_w__ls d_rch __n L__rz__ch_n g_tr_nnt __sg_g_b_n w_rd_n. D_s _bschl__ss_nd_ W_rt "\ST_P" s_ll n_cht __sg_g_b_n w_rd_n.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **4. Übungsblatt**

Musterlösung

Aufgabe 4.1

a) Annahme: Die Bezeichner werden nur aus Klein- und Großbuchstaben gebildet.

N1 = "aaa" .
N2 = "aab" .
N3 = "aac" .
...
N26 = "aaz" .
...
N52^{3} = "ZZZ" .

PROC = "PROCEDURE" **N1 Rest** "END" **N1** ";" |
"PROCEDURE" **N2 Rest** "END" **N2** ";" |
"PROCEDURE" **N3 Rest** "END" **N3** ";" |
...
"PROCEDURE" **N52**^{3} **Rest** "END" **N52**^{3} ";" .
Rest = ...

Wenn man die Lösung sieht, weiß man, warum man solche Zusammenhänge umgangssprachlich beschreibt.

b) **D_NUMERAL** = **D** | {["_"] **D**} .
D = "0" | ... | "9" .

Oder

D_NUMERAL = { "0" | ... | "9" | "_" }

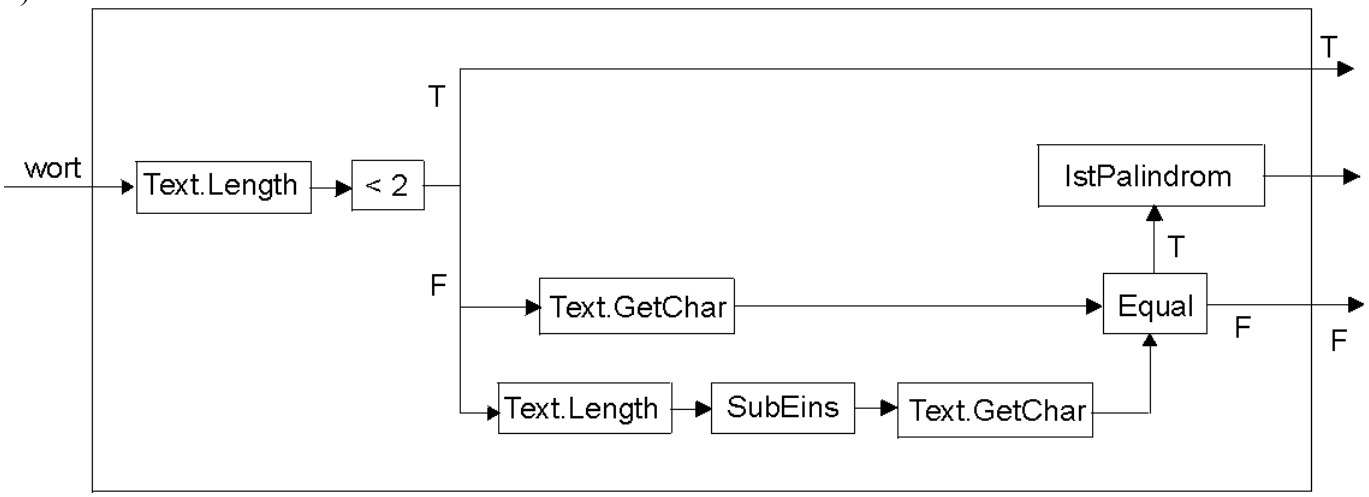
Die Idee ist mit dem dem Unterstrich ("_") den Trennpunkt bei Zahlen zu simulieren, also z.B. 1.000 durch 1_000.

Aufgabe 4.2

a)

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 4.2b

  Modul:   Palindrom
  Datei:   Palindrom.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
MODULE Palindrom EXPORTS Main;

IMPORT SIO,Text;

(*-------------------------------------------------------------------------
  Name:      IstPalindrom

  Aufgabe:   Die Funktion ueberprueft, ob das Wort, das als Parameter der
             Funktion uebergeben wurde, ein Palindrom ist. Dazu vergleicht
             die Funktion den ersten und den letzten Buchstaben des Wortes.
             Sind beide gleich, ruft sich die Funktion fuer den zwischen dem
             ersten und dem letzten Buchstaben liegenden Rest des Textes
             rekursiv auf um zu testen, ob es sich dabei ebenfalls um ein
             Palindrom handelt. Die Rekursion endet, wenn der Rest des
             Wortes weniger als 2 Zeichen enthaelt. In diesem Fall muss man
             nichts mehr ueberpruefen, sondern kann sofort TRUE zurueckgeben.

  Parameter: wort - das Wort, fuer das die Palindromeigenschaft getestet
                    werden soll

  Rueckgabe: TRUE wenn das Wort ein Palindrom ist, FALSE wenn nicht
-------------------------------------------------------------------------*)
PROCEDURE IstPalindrom(wort:TEXT): BOOLEAN =
  BEGIN
    IF Text.Length(wort) < 2 THEN (* Hat das Wort weniger als 2 Zeichen, *)
      RETURN TRUE;                 (* dann ist es ein Palindrom.          *)
    ELSE
         (* Teste, ob der erste und der letzte Buchstabe gleich sind. *)

      IF Text.GetChar(wort,0) = Text.GetChar(wort,Text.Length(wort)-1) THEN

         (* Wenn erster und letzter Buchstabe gleich sind, fuehre den
            selben Test fuer den Text dazwischen aus *)

        RETURN IstPalindrom(Text.Sub(wort,1,Text.Length(wort)-2));
      ELSE
        RETURN FALSE; (* erster und letzter ungleich -> kein Palindrom *)
      END;
    END;
  END IstPalindrom;
(*-------------------------------------------------------------------------
  Name:      TestePalindrom

  Aufgabe:   Die Prozedur erhaelt als Eingabeparameter ein wort. Sie benutzt
             die Funktion IstPalindrom, um zu ermitteln, ob es sich dabei um
             ein Palindrom handelt oder nicht. Das Ergebnis des Tests gibt
             die Prozedur auf dem Bildschirm aus.

  Parameter: wort - das zu testende Wort

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE TestePalindrom(wort:TEXT) =
  BEGIN
    IF IstPalindrom(wort) THEN
      SIO.PutText ("Das Wort " & wort & " ist ein Palindrom.");
    ELSE
      SIO.PutText ("Das Wort " & wort & " ist kein Palindrom.");
    END;
    SIO.Nl();
  END TestePalindrom;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
  SIO.PutText("Geben Sie bitte ein Wort ein: ");
  TestePalindrom(SIO.GetLine());
END Palindrom.
```

b)



Aufgabe 4.3

```
(*-------------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 4.3

  Modul:   Sinus
  Datei:   Sinus.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------------*)
MODULE Sinus EXPORTS Main;

IMPORT SIO,Fmt;
(*-------------------------------------------------------------------------------
  Name:      Pow

  Aufgabe:   Die Funktion erhaelt als Eingabeparameter zwei Zahlen x und n, wobei
             n positiv und ganzzahlig ist. Die Funktion liefert als Ergebnis
             die n-te Potenz der Zahl x. Die Funktion arbeitet rekursiv und
             nutzt dabei aus, dass x^n = x * x^(n-1) ist.

  Parameter: x - die zu potenzierende Zahl
             n - der Exponent

  Rueckgabe: x hoch n
-------------------------------------------------------------------------------*)
PROCEDURE Pow(x:LONGREAL; n:CARDINAL): LONGREAL =
  BEGIN
    IF n = 0 THEN
      RETURN 1.0D0;                        (* wenn n=0 ist, ist das Ergebnis 1 *)
    ELSE
      RETURN x * Pow(x,n-1);     (* sonst berechne x * x^(n-1) *)
    END;
  END Pow;

(*-------------------------------------------------------------------------------
  Name:      Fac

  Aufgabe:   Die Funktion berechnet die Fakultaet fuer eine positive
             ganze Zahl x (siehe auch Skript zur Vorlesung).

  Parameter: x - die Zahl, deren Fakultaet berechnet werden soll

  Rueckgabe: x!
-------------------------------------------------------------------------------*)
PROCEDURE Fac(x:CARDINAL): LONGREAL =
  BEGIN
    IF x<2 THEN RETURN 1.0D0
    ELSE RETURN FLOAT(x,LONGREAL) * Fac(x-1);
    END;
  END Fac;
```

```
(*-------------------------------------------------------------------------
  Name:       Sum

  Aufgabe:    Die Funktion summiert die einzelnen Terme der Potenzreihe auf,
              durch die der Sinus berechnet wird. Als erstes wird ueberprueft,
              ob der Betrag des naechsten zu addierenden Terms groesser als
              10^-9 ist. Je groesser n wird, desto kleiner werden die
              Betraege der Terme. Sie veraendern daher immer weniger am
              Gesamtergebnis. Man sucht sich deshalbe vorher eine
              Genauigkeitsschwelle aus, bis zu der man das Ergebnis berechnen
              will. In der Aufgabenstellung war diese Schwelle als 10^9
              vorgegeben.
              Falls der Betrag des naechsten Terms noch ueber dieser Schwelle
              liegt, wird die Summe aus diesem Term und dem Rest der Reihe
              gebildet. Den Wert der restlichen Reihe erhaelt man durch
              den rekursiven Aufruf von Sum.
              Falls der Betrag des naechsten Terms unterhalb der Schwelle liegt,
              wird als Ergebnis 0 zurueckgegeben und die Rekursion bricht an
              dieser Stelle ab.

  Parameter:  x - die Zahl, fuer die Summe der Reihe berechnet werden soll
              n - der Index ab dem die Reihe aufaddiert werden soll
              (siehe Aufgabenstellung)

  Rueckgabe: der Wert der Reihe fuer die Zahl x ab dem Index n
-------------------------------------------------------------------------*)
PROCEDURE Sum(x:LONGREAL; n:CARDINAL): LONGREAL =
  BEGIN
    (* Teste, ob der naechste Term groesser als der Schwellenwert ist. *)

    IF Pow(x,2*n+1) / Fac(2*n+1) > 0.000000001D0 AND THEN

      (* Wenn der Wert groesser ist, bilde die Summe aus
             |<-------- diesem Element der Reihe --->| |<- und den folgenden ->|*)
      RETURN Pow(-1.0D0,n) * Pow(x,2*n+1) / Fac(2*n+1)+        Sum(x,n+1);
    ELSE
      RETURN 0.0D0; (* wenn der Wert unter der Schwelle liegt, gibt 0 zurueck *)
    END;
  END Sum;
(*-------------------------------------------------------------------------
  Name:       SinCos

  Aufgabe:    Die Prozedur berechnet fuer einen Wert x den Sinus un den
              Cosinus und gibt diese Werte auf dem Bildschirm aus. Sie
              verwendet fuer diese Berechnung die Funktion Sum und nutzt
              fuer die Berechnung des Cosinus aus, dass
              cos(x) = sin(PI/2 + x) gilt.

  Parameter:  x - die Zahl, deren Sinus und Cosinus berechnet werden soll.

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE SinCos(x:LONGREAL) =
  BEGIN
    SIO.Nl();
    SIO.PutText("Sin(" & Fmt.LongReal(x) & ") =" & Fmt.LongReal(Sum(x,0)));
    SIO.Nl();
    SIO.PutText("Cos(" & Fmt.LongReal(x) & ") =" & Fmt.LongReal(Sum(1.5707963267949D0 +
x,0)));
    SIO.Nl();
  END SinCos;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
  SIO.PutText("Geben Sie einen Wert fuer x ein: ");
  SinCos(SIO.GetLongReal());
END Sinus.
```

Aufgabe 4.4

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung 1 - WS 1999/2000
  Loesung der Uebungsaufgabe 4.4

  Modul:   Xmas
  Datei:   Xmas.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE Xmas EXPORTS Main;

IMPORT SIO,Fmt;

(*---------------------------------------------------------------------------
  Name:      Heiligabend

  Aufgabe:   Die Prozedur erhaelt als Parameter eine Jahreszahl groesser 1582
             und gibt auf dem Bildschirm aus, auf welchen Wochentag der
             Heiligabend in jenem Jahr faellt.

  Parameter: jahr

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE Heiligabend (jahr: CARDINAL) =
VAR
  wochentagNr:CARDINAL;

  BEGIN
    IF jahr < 1582 THEN
      SIO.PutText("Tut mir leid, an das Jahr kann ich mich nicht mehr erinnern!");
    ELSE
      SIO.PutText("Der Heiligabend im Jahr " & Fmt.Int(jahr) & " ist ein ");

      (* Initialisierung auf Freitag fuer das Jahr 1582 *)

      wochentagNr := 5;
      jahr := jahr - 1582;

      (* Wochentag pro Jahr um 1, pro Schaltjahr um 2 weitergehen.
         Berechnung der Zahl der Schaltjahre nach den Regeln in der
         Aufgabenstellung *)

      wochentagNr := (wochentagNr
                      +  (jahr MOD 7)         (* Ueberlauf verhindern *)
                      +  ((jahr+2 )  DIV 4)
                      -  ((jahr+82)  DIV 100)
                      +  ((jahr+382) DIV 400)) MOD 7;

      CASE wochentagNr OF
      | 0 => SIO.PutText("Sonntag.");
      | 1 => SIO.PutText("Montag.");
      | 2 => SIO.PutText("Dienstag.");
      | 3 => SIO.PutText("Mittwoch.");
      | 4 => SIO.PutText("Donnerstag.");
      | 5 => SIO.PutText("Freitag.");
      | 6 => SIO.PutText("Samstag.");
      END;
    END;
    SIO.Nl();
  END Heiligabend;
(*---------------------------------------------------------------------------
  Hauptprogramm
---------------------------------------------------------------------------*)
BEGIN
  SIO.PutText("Bitte geben Sie eine Jahreszahl >= 1582 ein :");
  Heiligabend(SIO.GetInt());
END Xmas.
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **5. Übungsblatt**

Musterlösung

Aufgabe 5.1

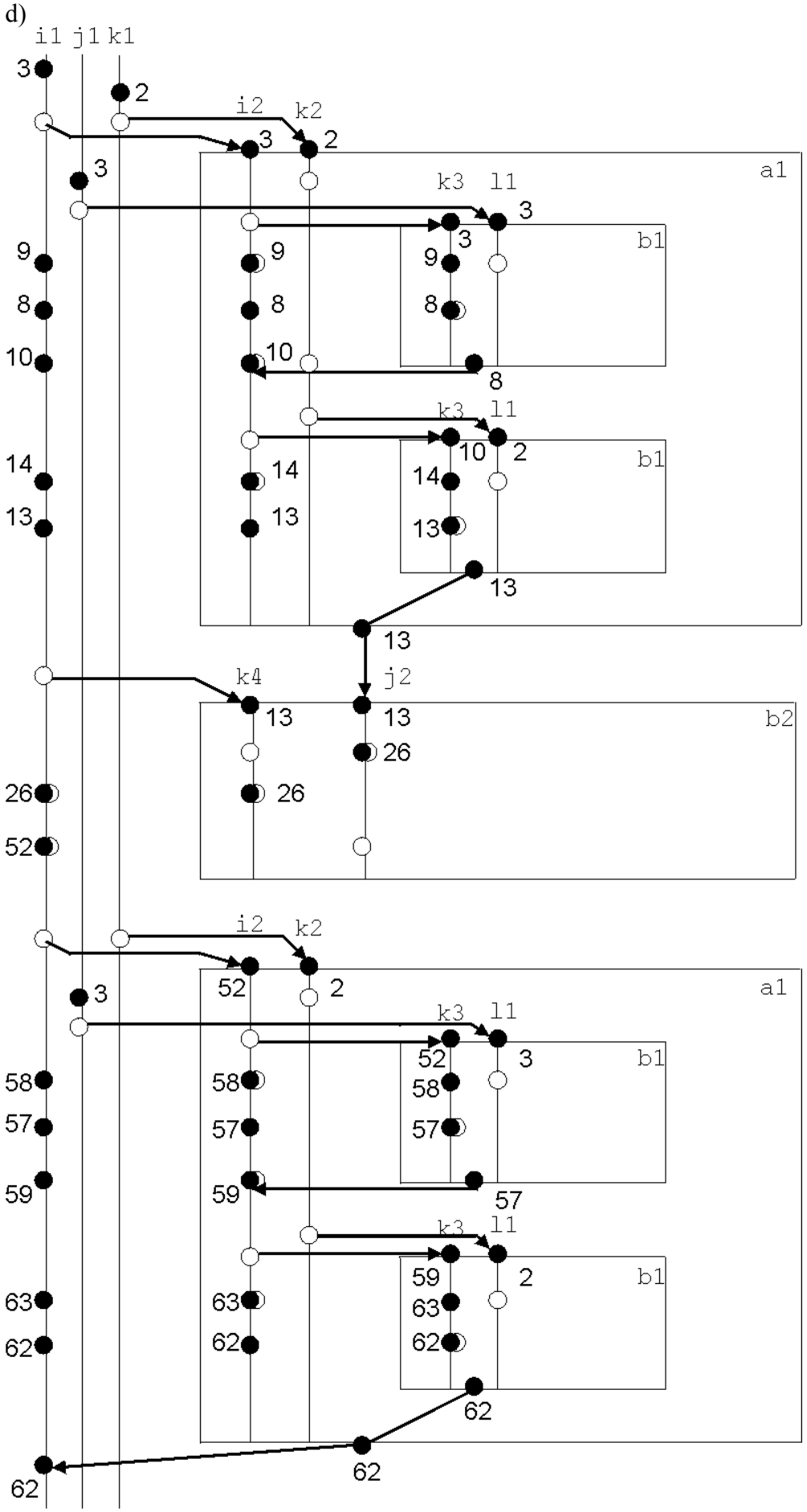
a)

```
01 MODULE M EXPORTS Main;
02
03 VAR i1, j1, k1 : INTEGER;
04
05 PROCEDURE a1 (VAR i2, k2 : INTEGER): INTEGER =
06
07     PROCEDURE b1(VAR k3 : INTEGER; l1 : INTEGER): INTEGER =
08     BEGIN
09       i2 := i2 + 2 * l1;
10       k3 := k3 - 1;
11       RETURN i2;
12     END b1;
13
14 BEGIN
15   j1 := k2 + 1 ;
16   i2 := k2 + b1(i2, j1);
17   RETURN b1(i2, k2);
18 END a1;
19
20
21 PROCEDURE b2 (j2 : INTEGER; VAR k4 : INTEGER) =
22 BEGIN
23   j2 := j2 + k4;
24   k4 := k4 + i1;
25   i1 := i1 + j2;
26 END b2;
27
28 BEGIN
29   i1 := 3;
30   k1 := 2;
31   b2( a1(i1, k1), i1);
32   i1 := a1( i1, k1)
33 END M.
```

b) `i` aus Zeile 03 ist ab Zeile 21 bis Zeile 33 sichtbar
`k` aus Zeile 03 ist ab Zeile 28 bis Zeile 33 sichtbar
`k` aus Zeile 05 ist ab Zeile 14 bis Zeile 18 sichtbar
`k` aus Zeile 07 ist ab Zeile 08 bis Zeile 12 sichtbar
`j` aus Zeile 21 ist ab Zeile 22 bis Zeile 26 sichtbar

c) `k` aus Zeile 10 wird definiert in Zeile 07
`j` aus Zeile 15 wird definiert in Zeile 03
`i` aus Zeile 17 wird definiert in Zeile 05
`j` aus Zeile 23 wird definiert in Zeile 21
`i` aus Zeile 25 wird definiert in Zeile 03

d)


i1 j1 k1
3
2
i2 k2
3 2
a1
3
k3 l1
3
3
b1
9
9
9
8
8
8
10
10
8
k3 l1
10 2
14
14
14
13
13
13
13
13
k4
j2
13
13
b2
26
26
26
52
i2 k2
52
2
a1
3
k3 l1
52
3
b1
58
58
58
57
57
57
59
59
57
k3 l1
59
2
b1
63
63
63
62
62
62
62
62

Aufgabe 5.2

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 5.2

  Modul:   Segment
  Datei:   Segment.m3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
MODULE Segment EXPORTS Main;

FROM Math IMPORT log10, pow;
FROM SIO  IMPORT PutText, PutInt, GetInt;

TYPE DigitType = [0..9];
VAR Num: CARDINAL;
(*-----------------------------------------------------------------------------
  Name:     SegmentCode

  Aufgabe:  Gibt zu eine gegebenen positiven ganzen Zahl die Folge von
            Segmenten eine 7-Segmentanzeige aus, mit denen diese Zahl
            dargestellt wird.
            Es waren prinzipiell zwei Loesungswege moeglich, einer, bei dem
            die Zahl als String verarbeitet wird und einer bei dem die
            Zahl als CARDINAL verarbeitet wird. Dieses Programm implementiert
            den zweiten Loesungsweg. Ausserdem war durch die Einschraenkung,
            dass die Zahl maximal fuenf Stellen haben soll, eine einfachere
            Loesung moeglich als die hier implementierte, die fuer eine
            beliebige Anzahl von Stellen funktioniert. (Natuerlich nur,
            solange der Wertebereich von CARDINAL nicht ueberschritten wird.)

  Parameter: number - die Zahl, fuer die die Segmente auszugeben sind

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE SegmentCode(number:CARDINAL) =
  VAR s:DigitType;
      t:CARDINAL;
      max:CARDINAL;
  BEGIN

    (* Zuerst berechnet man die hoechste Stelle die bei der Zahl belegt
       ist. Bei einstelligen Zahlen ist dies die Stelle 10 hoch 0,
       bei zweistelligen 10 hoch 1, bei dreistelligen 10 hoch 2 usw.
       Der folgenden Ausdruck berechnet genau diesen Exponenten (0,1,2,usw.)
       und legt in in der Variablen max ab. *)

    max:=FLOOR(log10(FLOAT(number,LONGREAL)));

    (* Nun wissen wir, wieviele Stellen die Zahl hat. Fuer jede dieser
       Stellen durchlaufen wir folgende Schleife: *)

    FOR i:=max TO 0 BY -1 DO

      (* Dieser Ausdruch berechnet aus dem i (der Schleifenvariablen)
         wieder die Zehnerpotenz. Eigentlich steht hier  t := 10 hoch i.
         Kompliziert sieht es nur durch die ganzen Typkonvertierungen aus. *)

      t := ROUND(pow(10.0D0,FLOAT(i,LONGREAL)));

      (* Wir dividieren s durch t und erhalten so die Wertigkeit der
         vordersten Stelle der Zahl. Bsp.: 4567 DIV 1000 = 4 *)

      s := number DIV t;

      (* Mit der berechneten Wertigkeit wissen wir nun, welche Segmente
         angeschaltet sind und geben diese aus. *)

      CASE s OF
      | 0 => PutText("0: A B C   E F G\n");
      | 1 => PutText("1:     C     F  \n");
      | 2 => PutText("2: A   C D E   G\n");
      | 3 => PutText("3: A   C D   F G\n");
      | 4 => PutText("4:  B C D   F  \n");
      | 5 => PutText("5: A B   D   F G\n");
      | 6 => PutText("6: A B   D E F G\n");
      | 7 => PutText("7: A   C     F  \n");
      | 8 => PutText("8: A B C D E F G\n");
      | 9 => PutText("9: A B C D   F G\n");
      END;

      (* Zuletzt ziehen wir die vorderste Stelle, fuer die wir das Segment
         gerade ausgegeben haben, von der Zahl ab. Damit ist die naechst
         kleinere Stelle dir neue vorderste Stelle und das Verfahren
         funktioniert im naechsten Schleifendurchlauf wieder genauso. *)

      number := number - s * t;
    END;
  END SegmentCode;
```

```
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
 PutText("Bitte geben Sie eine Zahl ein: ");
 Num := GetInt();
 PutText("Die Zahl "); PutInt(Num); PutText(" wird angezeigt durch die Segmente:\n");
 SegmentCode(Num);
END Segment.
```

Aufgabe 5.3

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 5.3

  Modul:   TimeCalc
  Datei:   TimeCalc.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
MODULE TimeCalc EXPORTS Main;

FROM SIO IMPORT PutText, GetInt, PutInt, Nl;

VAR Hrs,Min,dHrs,dMin,newHrs,newMin:CARDINAL;
(*-------------------------------------------------------------------------
  Name:      AddTime

  Aufgabe:   Die Prozedur addiert zu einer Uhrzeit eine Zeitspanne und
             liefert als Ergebnis die Uhrzeit nach dem Verstreichen dieser
             Zeitspanne.

  Parameter: hours         - Stunden der aktuellen Uhrzeit
             minutes       - Minuten der aktuellen Uhrzeit
             deltaHours    - Stunden der Zeitspanne
             deltaMinutes  - Minuten der Zeitspanne
             resultHoures  - Rueckgabeparameter: Hier stehen nach Ende der
                             Prozedur die Stunden der Ergebnisuhrzeit.
             resultMinutes - Rueckgabeparameter: Hier stehen nach Ende der
                             Prozedur die Minuten der Ergebnisuhrzeit.

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE AddTime (hours, minutes, deltaHours, deltaMinutes: CARDINAL;
           VAR resultHours, resultMinutes: CARDINAL) =

  BEGIN

    (* Wenn die eingegebenen Minuten groesser oder gleich 60 sind, werden die
       Stunden entsprechend erhoeht *)
    deltaHours   := deltaHours + deltaMinutes DIV 60;

    (* Anschliessend werden alle Minuten groesser oder gleich 60 gekappt *)
    deltaMinutes := deltaMinutes MOD 60;

    (* Die Minuten des Ergebnisses ergeben sich aus den Minuten der Uhrzeit
       plus den zu addierenden Minuten. Werte groesser gleich 60 werden
       hier wieder durch MOD gekappt *)
    resultMinutes:=(minutes + deltaMinutes) MOD 60;

    (* Die Stunden des Ergebnisses ergeben sich aus den Stunden der Uhrzeit
       plus den bei der Berechnung der Minuten gekappten Stunden plus
       den Stunden die zu addieren sind. Auch hier werden Stunden groesser
       oder gleich 24 durch MOD gekappt. (Es war in der Aufgabe ja nicht
       verlangt, dass man ausgibt, bei wieviel Tagen spaeter man landet,
       sondern nur, dass man sagt wie spaet es dann ist. *)
    resultHours  :=(hours + (minutes + deltaMinutes) DIV 60 + deltaHours) MOD 24;
  END AddTime;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
  PutText("Bitte geben Sie die Stunden der Uhrzeit ein : ");
  Hrs := GetInt();
  PutText("Bitte geben Sie die Minuten der Uhrzeit ein : ");
  Min := GetInt();
  PutText("Wieviele Stunden weiter                    : ");
  dHrs := GetInt();
  PutText("Wieviele Minuten weiter                    : ");
  dMin := GetInt();

  AddTime (Hrs,Min,dHrs,dMin,newHrs,newMin);
  Nl();
  PutText("In ");  PutInt(dHrs);  PutText(" h und ");  PutInt(dMin);
  PutText(" min von ");  PutInt(Hrs);  PutText(".");  PutInt(Min);
  PutText(" Uhr an gerechnet ist es ");  PutInt(newHrs);  PutText(".");
  PutInt(newMin);  PutText(" Uhr.");  Nl();
END TimeCalc.
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **6. Übungsblatt**

Musterlösung

Aufgabe 6.1

```
(*-------------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 3.4

  Modul:   Replace2
  Datei:   Replace2.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------------*)
MODULE Replace2 EXPORTS Main;

FROM Stdio IMPORT stdout,stdin;
FROM Wr IMPORT PutText;
FROM Rd IMPORT GetLine;
FROM Text IMPORT Length, GetChar, Sub, Equal, FromChar, Empty;

(*-------------------------------------------------------------------------------
  Name:       InputUntilStop

  Aufgabe:    Die Funktion liest einen Text zeilenweise ein und verbindet
              die Zeilen zu einem einzigen Text. Wenn das Wort "\STOP"
              einzeln  in einer Zeile eingelesen wird, wird die
              Funktion beendet.

  Parameter:  lines      - die Zeilen, die bereits gelesen wurden
              newLine    - die zuletzt vom Benutzer eingegebene Zeile

  Rueckgabe:  ein Texte, der die vom Benutzer eingegeben Zeilen, getrennt
              jeweils durch ein Newline Zeichen, enthaelt
-------------------------------------------------------------------------------*)

PROCEDURE InputUntilStop ():TEXT =
  VAR lines:TEXT;
      newLine:TEXT;
  BEGIN
    lines:="";
    LOOP
      newLine := GetLine(stdin);
      IF Equal(newLine,"\\STOP") THEN EXIT;
      ELSE
        lines:= lines & newLine;
      END;
    END;
    RETURN lines;
  END InputUntilStop;


(*-------------------------------------------------------------------------------
  Name:       ReplaceVowels

  Aufgabe:    Die Funktion erhaelt als Eingabe einen Text im Parameter
              text. Sie durchlaueft diesen Text zeichenweise. Wenn das
              zuletzt gelesene Zeichen kein Vokal ist, wird es zum
              Ergebnistext hinzugefuegt, wenn es ein Vokal ist, wird
              stattdessen ein Unterstrich angefuegt. Der aus den
              gelesenen Zeichen und Unterstrichen zusammengefuegte
              Text wird in der lokalen Variable result gepeichert.
              Ist der gesamte Text durchlaufen, wird der Inhalt von
              result zurueckgegeben.

  Parameter:  text - der zu bearbeitende Text

  Rueckgabe:  der Inhalt von text, wobei alle Vokale durch
              Unterstriche '_' ersetzt wurden.
-------------------------------------------------------------------------------*)
```

```
PROCEDURE ReplaceVowels(text:TEXT):TEXT =

  VAR result:TEXT;     (* Zwischenspeicher fuer das Ergebnis der Funktion *)
      character:CHAR;  (* das aktuell bearbeitete Zeichen *)

  BEGIN
    FOR i:=0 TO Length(text) DO
      character:= GetChar(text,i);
      CASE character OF
      | 'e','E','o','O','a','A','i','I','u','U' =>
        result:= result & "_";
      ELSE
        result := result & FromChar(character);
      END
    END;
    RETURN result;
END ReplaceVowels;

(*-------------------------------------------------------------------------------
  Hauptprogramm

  Hier wird die Funktion PutText verwendet, um das Ergebnis der Funktion
  ReplaceVowels auszugeben, die wiederum auf dem Ergebnis der Funktion
  InputUntilStop arbeitet. InputUntilStop wird mit zwei leeren Strings
  als aktuellen Parametern aufgerufen, da es zu diesem Zeitpunkt noch
  keine bisher eingegebene Zeilen (erster Parameter) und keine neue Zeile
  (zweiter Paremeter) gibt.
-------------------------------------------------------------------------------*)
BEGIN
  PutText(stdout, ReplaceVowels(InputUntilStop()));
END Replace2.
(*-------------------------------------------------------------------------------
 PROGRAMMENDE
-------------------------------------------------------------------------------*)
```

Aufgabe 6.2

a)

```
PROCEDURE Potenz_mit_FOR ( a, b: INTEGER ) : INTEGER;
VAR i, resultat: INTEGER;
BEGIN
      resultat := 1;
FOR i:= 1 TO b DO
            resultat := resultat * a;
      END (* FOR *)
      RETURN resultat
END Potenz_mit_FOR;


PROCEDURE Potenz_mit_WHILE ( a, b: INTEGER ) : INTEGER;
VAR i, resultat: INTEGER;
BEGIN
      i := 1;
resultat := 1;
WHILE i <= b
            resultat := resultat * a;
            INC(i);
      END (* WHILE *)
      RETURN resultat
END Potenz_mit_WHILE;


PROCEDURE Potenz_mit_REPEAT ( a, b: INTEGER ) : INTEGER;
VAR i, resultat: INTEGER;
BEGIN
      IF b = 0; THEN
            RETURN 1
      ELSE
            i := 0;
            resultat := 1;
REPEAT
                  resultat := resultat * a;
                  INC(i);
            UNTIL i=b;
RETURN resultat
      END (* IF *)
END Potenz_mit_REPEAT;


PROCEDURE Potenz_mit_LOOP ( a, b: INTEGER ) : INTEGER;
VAR i, resultat: INTEGER;
BEGIN
      i := 1;
resultat := 1;
LOOP
      IF i>b THEN EXIT END;
            resultat := resultat * a;
            INC(i);
      END (* LOOP *)
      RETURN resultat
END Potenz_mit_LOOP;
```

b)
Die FOR-Schleife führt bei diesem Anwendungsfall zur einfachsten Lösung, darum ist sie in diesem Fall vorzuziehen, hierbei treten keine Nachteile auf.

Die Lösung mit WHILE- und LOOP-Schleife ähneln sich und sind beide tragbar, falls aus irgendwelchen Gründen keine FOR-Schleife möglich ist. Die boolesche Bedingung ist eigentlich überflüssig, weil die Anzahl Schleifendurchläufe bekannt ist.

Die REPEAT ist klar die umständlichste Konstruktion, weil der Test vor dem ersten Durchlauf nicht entfallen kann. Die Lösung könnte Vorteile bringen in einer Umgebung, in der b oft den Wert 0 hat und es auf Effizienz ankommt. Man hat in diesem speziellen Fall ein ausgezeichnetes Zeitverhalten.

Aufgabe 6.3

```
(*-------------------------------------------------------------------------------
  Lehrstuhl für Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 6.3

  Modul  :   Kraftstoffverbrauch
  Datei  :   Kraftstoffverbrauch.m3
  Autorin:   Katja Cremer
-------------------------------------------------------------------------------*)
MODULE Kraftstoffverbrauch EXPORTS Main;

FROM SIO IMPORT PutText, Nl, GetInt, PutReal, GetChar;
FROM Math IMPORT log;

(* Definition eines Aufzählungsdatentyps, der die möglichen Fahrzeugarten umfaßt   *)
TYPE Fahrzeugart  = {Fahrrad, Skateboard, Mofa, Moped,
                     Motorrad, Pkw, Lkw, Bus};

(* Definition eines Verbunddatentyps, der die Fahrzeugart und den errechneten
   Verbrauch aufnimmt *)
Verbrauch   =   RECORD
                    Fahrzeug    : Fahrzeugart;
                    Menge       : REAL;
                END;


VAR KVerbrauch  :   Verbrauch;
    Auswahl     :   CHAR;
    Eingabe     :   INTEGER;


BEGIN

(* Zunächst erfolgt die Auswahl eines Fahrzeugs*)

(* Ausgabe der möglichen Fahrzeugarten *)
PutText("Fahrzeugtyp angegeben: (F)ahrrad, (S)kateboard, (M)otorrad"); Nl();
PutText("                       M(o)fa, Mop(e)d, (P)kw, (L)kw, (B)us"); Nl(); Nl();

PutText("Auswahl (in Kleinbuchstaben): ");

Auswahl  := GetChar(); Nl();

(* Je nach Auswahl werden in der CASE-Anweisung weitere Informationen vom Benutzer
   angefordert und daraus der Verbrauch errechnet*)

CASE Auswahl OF
    'f', 's'        =>  IF Auswahl = 'f' THEN
                            KVerbrauch.Fahrzeug := Fahrzeugart.Fahrrad;
                        ELSE
                            KVerbrauch.Fahrzeug := Fahrzeugart.Skateboard;
                        END;

                        PutText("Körpergewicht der Person (in kg): ");
                        Eingabe := GetInt();
                        KVerbrauch.Menge := 0.1 * FLOAT((Eingabe - 1) DIV 10 + 1);
```

```
    | 'o', 'e', 'm' =>  CASE Auswahl OF
                            'o'   =>  KVerbrauch.Fahrzeug := Fahrzeugart.Mofa;
                          | 'e'   =>  KVerbrauch.Fahrzeug := Fahrzeugart.Moped;
                          | 'm'   =>  KVerbrauch.Fahrzeug := Fahrzeugart.Motorrad;
                        END;

                        PutText("Körpergewicht der Person (in kg): ");
                        Eingabe := GetInt();
                        KVerbrauch.Menge := 0.1 * FLOAT((Eingabe - 1) DIV 10 + 1) + 4.0;


    | 'p'           =>  KVerbrauch.Fahrzeug := Fahrzeugart.Pkw;
                        PutText("Anzahl der Insassen: ");
                        Eingabe := GetInt();
                        KVerbrauch.Menge := 6.0 + (0.5 * FLOAT(Eingabe));


    | 'l'           =>  KVerbrauch.Fahrzeug := Fahrzeugart.Lkw;
                        PutText("Zuladung (in t): ");
                        Eingabe := GetInt();
                        KVerbrauch.Menge := 15.0 + (1.5 *
FLOAT(log(FLOAT(FLOAT(Eingabe),LONGREAL))));

    | 'b'           =>  KVerbrauch.Fahrzeug := Fahrzeugart.Bus;
                        PutText("Anzahl der Insassen: ");
                        Eingabe := GetInt();
                        KVerbrauch.Menge := 17.0 + (1.3 * FLOAT((Eingabe - 1) DIV 12 + 1 ));

END;

(* Ausgabe der Ergebnisse *)

PutText("Kraftstoffverbrauch: ");
PutReal(KVerbrauch.Menge);
PutText(" Liter pro 100km"); Nl();

END Kraftstoffverbrauch.
```

Aufgabe 6.4

```
(*-------------------------------------------------------------------------------
  Lehrstuhl für Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 6.4

  Modul  :   Fahrzeugschein
  Datei  :   Fahrzeugschein.m3
  Autorin:   Katja Cremer
-------------------------------------------------------------------------------*)
MODULE Fahrzeugschein EXPORTS Main;

FROM SIO IMPORT PutText, GetInt, GetChar, GetReal, GetLine, Nl;


(* Typdefinitionen für das Kennzeichen und die Fahrzeugmasse *)
TYPE
Kennzeichen = RECORD
                Ort        : ARRAY [1..3] OF CHAR;
                Buchstaben : ARRAY [1..3] OF CHAR;
                Zahlen     : ARRAY [1..3] OF CHAR;
              END;

Masse = RECORD
          Laenge : REAL;
          Breite : REAL;
          Hoehe  : REAL;
        END;


(* der eigentliche Datentyp für den Fahrzeugschein *)
Fahrzeugschein = RECORD
                   Herstellerfirma      : TEXT;
                   Fahrzeugkennzeichen  : Kennzeichen;
                   Leistung             : CARDINAL;
                   zulaessiges_GGewicht : REAL;
                   Hersteller_Schalld   : TEXT;
                   Zulassungsnr_Schalld : CARDINAL;
                   Fahrzeugmasse        : Masse;
                   Achsenanzahl         : CARDINAL;
                   Nutzlast             : REAL;
                   Tankinhalt           : REAL;
                 END;


VAR Schein : Fahrzeugschein;
    Auswahl: CARDINAL;
```

```
(*-------------------------------------------------------------------------
  Name:       EinlesenFahrzeugschein

  Aufgabe:    Die Prozedur realisiert eine einfache Benutzerschnittstelle zur
              Eingabe der Fahrzeugdaten, die in der Datenstruktur Fahrzeugschein
              gespeichert werden. Dazu werden in die Prozedur die notwendigen
              Eingaben vom Benutzer angefordert.

  Parameter: Schein, Referenzvariable von Typ Fahrzeugschein

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE EinlesenFahrzeugschein (VAR Schein: Fahrzeugschein) =
BEGIN

(* Zuerst werden die Daten abgefragt, die fuer alle Fahrzeugarten erfaßt
   werden muessen *)

PutText("Einlesen eines Fahrzeugscheins"); Nl();
PutText("Allgemeine Daten"); Nl(); Nl();

PutText("Herstellerfirma: ");
Schein.Herstellerfirma := GetLine();

PutText("Ortscode des Kennzeichen: ");
FOR i:= 1 TO 3 DO
  Schein.Fahrzeugkennzeichen.Ort[i] := GetChar();
END;
Nl();

PutText("Buchstaben des Kennzeichen: ");
FOR i:= 1 TO 3 DO
  Schein.Fahrzeugkennzeichen.Buchstaben[i] := GetChar();
END;

PutText("Nummer des Kennzeichen: ");
FOR i:= 1 TO 3 DO
  Schein.Fahrzeugkennzeichen.Zahlen[i] := GetChar();
END;

PutText("Leistung (in PS): ");
Schein.Leistung := GetInt(); Nl();

PutText("Zulässiges Gesamtgewicht (in kg): ");
Schein.zulaessiges_GGewicht := GetReal(); Nl();


(* Erfassung der Daten, die nur einige Fahrzeugarten relevant sind *)

PutText("Spezielle Daten"); Nl();

PutText("Motorrad=1, Pkw=2 oder Lkw=3, Bitte auswählen"); Nl();
Auswahl := GetInt();


CASE Auswahl OF
(* Daten fuer Motorraeder *)
  1        =>     PutText("Hersteller des Schalldämpfers: ");
                  Schein.Hersteller_Schalld := GetLine(); Nl();
                  PutText("Zulassungsnummer: ");
                  Schein.Zulassungsnr_Schalld := GetInt(); Nl();

(* Daten fuer PKWs und LKWs *)
| 2,3      =>     PutText("Länge (in cm): ");
                  Schein.Fahrzeugmasse.Laenge := GetReal(); Nl();
                  PutText("Höhe: (in cm)");
                  Schein.Fahrzeugmasse.Hoehe := GetReal(); Nl();
                  PutText("Breite: (in cm)");
                  Schein.Fahrzeugmasse.Breite := GetReal(); Nl();
                  PutText("Zahl der Achsen: ");
                  Schein.Achsenanzahl := GetInt(); Nl();
                  IF Auswahl = 3 THEN
                    PutText("Nutzlast (in t): ");
                    Schein.Nutzlast := GetReal(); Nl();
                    PutText("Tankinhalt (in l): ");
                    Schein.Tankinhalt := GetReal(); Nl();
                  END;
END;

END EinlesenFahrzeugschein;


BEGIN

(* Aufruf der Prozedeur EinlesenFahrzeugschein im Hauptprogramm *)

EinlesenFahrzeugschein(Schein);

END Fahrzeugschein.
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **7. Übungsblatt**

Musterlösung

Aufgabe 7.1

```
(*-------------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 7.1

  Modul:   Heap
  Datei:   Heap.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------------*)
MODULE Heap EXPORTS Main;

IMPORT SIO,Fmt;

CONST
  MAXHEAP = 10;   (* Anzahl der Element und Index des letzten Elements *)

TYPE
  (* Zuerst definieren wir den Typ eines einzelnen Heap Elements *)

  Element = RECORD
    Free: BOOLEAN;   (* speichert die Information, ob dieses Feld frei ist.
                        Fuer das Funktionieren der Implementierung ist diese
                        Komponente nicht notwendig. Sie vereinfacht aber die
                        Prozedur PrintHeap.*)
    Data: INTEGER;   (* die Nutzdaten des Heap Elements *)
    Next: CARDINAL;  (* der Verweis auf das naechste Element der
                        Freispeicherliste *)
  END;

  (* Ein Heap besteht damit aus zwei Koponenten, naemlich dem Index des
     ersten freien Elements und dem Array, das die Elemente enthaelt. *)

  HeapType = RECORD
    First:    CARDINAL;                        (* erster freier Index *)
    Elements: ARRAY [1..MAXHEAP] OF Element;  (* Heap Elemente *)
  END;

VAR Heap: HeapType;      (* der Heap, mit dem wir im Hauptprogramm arbeiten *)

    ListStart,           (* der Start bzw. das Ende der Liste, die im *)
    ListEnd,             (* Hauptprogramm aufgebaut wird *)

    Aux :CARDINAL;       (* Ein Hilfsvariable, in der wir uns voruebergehend
                            den Index eines Listenelements merken koennen *)
(*-------------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Die Prozedur initialisiert den Heap. Das Feld First
             wird mit dem Wert 1 initialisiert, d.h. das erste freie Element
             des Heaps soll das Feld des Arrays mit dem Index 1 sein. Die
             Next Felder der Element des Arrays werden mit jeweils mit dem
             Index des naechsten Array Elements initialisiert. Das Feld Next
             des letzten Arraz Elements wird mit dem Wert 0 belegt. Dieser Wert
             markiert das Ende der Freispeicherliste.

  Parameter: heap - die Heap Datenstruktur, die initialisiert wird.

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE Init(VAR heap: HeapType) =
  BEGIN
    Heap.First := 1;
    FOR i:=1 TO MAXHEAP-1 DO
      heap.Elements[i].Next := i+1;
      heap.Elements[i].Free := TRUE;
    END;
    heap.Elements[MAXHEAP].Next := 0;
    heap.Elements[MAXHEAP].Free := TRUE;
  END Init;
```

```
(*-------------------------------------------------------------------------
  Name:      New

  Aufgabe:   Die Prozedur liefert den Index eines freien Elements im Heap
             zurueck, falls noch ein freies Element vorhanden ist.
             Das Element, dessen Index zurueckgeliefert wird, wird aus der
             Freispeicherliste herausgenommen.
             Falls kein Element mehr frei ist, wird der Wert 0 zurueckgegeben

  Parameter: heap - der Heap, aus dem ein freies Element entnommen wird

  Rueckgabe: falls noch mindestens ein Element frei ist: der Index eines
                                                         freien Elements.
             falls nicht:                                0
-------------------------------------------------------------------------*)
PROCEDURE New(VAR heap: HeapType):CARDINAL =
  VAR result:CARDINAL;
  BEGIN
    result:=heap.First;
    IF result # 0 THEN
      heap.First:=heap.Elements[result].Next;
      heap.Elements[result].Free := FALSE;
    END;
    RETURN result;
  END New;
(*-------------------------------------------------------------------------
  Name:      Dispose

  Aufgabe:   Die Prozedur gibt ein Element des Heaps, das vorher durch
             die Prozedur New belegt worden ist, wieder frei. Dazu wird
             der uebergebene Index des Elements wieder in die Freispeicherliste
             eingefuegt.

  Parameter: heap  - der Heap, in dem das Element wieder freigegeben werden soll
             index - der Index des Elements

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Dispose(VAR heap:HeapType; index:CARDINAL) =
  BEGIN
    IF index > 0 AND index <= MAXHEAP THEN
      heap.Elements[index].Next := heap.First;
      heap.Elements[index].Free := TRUE;
      heap.First := index;
    END;
  END Dispose;
(*-------------------------------------------------------------------------
  Name:      PrintHeap

  Aufgabe:   Die Prozedur gibt den Inhalt des Heaps auf dem Bildschirm aus.
             Die freien Felder werden durch ein vorangestelltes 'F' markiert.

  Parameter: heap - der Heap, der angezeigt werden soll

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintHeap (VAR heap: HeapType) =
  BEGIN
    SIO.PutText("-------------------------");SIO.Nl();
    SIO.PutText("First: " & Fmt.Int(heap.First));SIO.Nl();
    FOR i:=1 TO MAXHEAP DO
      IF heap.Elements[i].Free THEN
        SIO.PutText("F: ");
      ELSE
        SIO.PutText(" : ");
      END;
      SIO.PutText(Fmt.Int(i));
      SIO.PutText(" next: " & Fmt.Int(heap.Elements[i].Next));
      SIO.PutText(" data: " & Fmt.Int(heap.Elements[i].Data));
      SIO.Nl();
    END;
    SIO.PutText("-------------------------"); SIO.Nl();
  END PrintHeap;
```

```
(*-------------------------------------------------------------------------
  Hauptprogramm
  -------------------------------------------------------------------------*)
BEGIN
  Init(Heap);              (* Initialisieren des Heaps *)
  ListStart := 0;          (* ListStart wird mit 0 belegt -> Liste ist leer *)
  ListEnd   := 0;

  PrintHeap(Heap);         (* Inhalt des Heaps anzeigen *)

  ListStart := New(Heap);  (* zuerst ein Element belegen. Da dies im Moment das *)
  ListEnd   := ListStart;  (* einzige Element ist, ist es Anfang und Ende der Liste *)

  Heap.Elements[ListEnd].Next := 0;    (* Eine 0 im Feld Next markiert das Listenende *)
  Heap.Elements[ListEnd].Data := 1111; (* ein beliebigen Nutzdatum ablegen *)

  PrintHeap(Heap);         (* Inhalt des Heaps anzeigen *)

  (* Belegen der uebrigen Listenelemente in der folgenden Schleife: *)

  FOR i:=2 TO 4 DO
    Aux := New(Heap);                       (* neues Element belegen *)
    Heap.Elements[ListEnd].Next := Aux;     (* neues Element hinten an die *)
    ListEnd := Aux;                         (* Liste anhaengen *)
    Heap.Elements[ListEnd].Data := 1111*i; (* ein beliebigen Nutzdatum ablegen *)
    Heap.Elements[ListEnd].Next := 0;       (* 0 markiert das Listenende *)

    PrintHeap(Heap);                        (* Inhalt des Heaps anzeigen *)
  END;

  (* Und die Liste wieder freigeben: *)

  WHILE ListStart# 0 DO                    (* Wenn Start noch nicht wieder auf 0 *)
    Aux := Heap.Elements[ListStart].Next; (* zweites Element der Liste merken *)
    Dispose (Heap,ListStart);              (* erstes Element freigeben *)
    ListStart := Aux;                      (* das in Aux gemerkte Element wird *)
                                           (* das neue erste Element *)

    PrintHeap (Heap);                      (* Inhalt des Heaps anzeigen *)
  END;
END Heap.
```

Aufgabe 7.2

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 7.2

  Modul:   DoubleList
  Datei:   DoubleList.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
MODULE DoubleList EXPORTS Main;

IMPORT SIO, Text;

(* Diese 'Adressen' werden in das erste bzw. letzte Listenelement
   eingetragen. Diese Elemente markieren die beiden Enden der Liste. *)

CONST HeadElement = Address{"HEAD","---","---","---","---"};
      TailElement = Address{"TAIL","---","---","---","---"};


TYPE
  Address = RECORD   (* der Type der zu speichernden Adressen *)
    name:   TEXT;
    street: TEXT;
    num:    TEXT;
    post:   TEXT;
    city:   TEXT;
  END;

  Element = RECORD     (* der Typ eines Listenelements *)
    adr:  Address;
    next: REF Element;
    prev: REF Element;
  END;

  ListType = RECORD    (* der Typ der Liste *)
    head: REF Element; (* Zeiger auf das erste Element *)
    tail: REF Element; (* Zeiger auf das letzte Element *)
  END;

VAR
  List:ListType;
  input:TEXT;
```

```
(*---------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Die Prozedur initialisiert die Liste. Damit man beim Einfuegen
             und Loeschen nicht die Sonderfaelle (leere Liste, einfuegen am
             Ende, einfuegen am Anfang) behandeln muss, wird die Liste
             initial mit einem vordersten Element (head) und einem
             hintersten Element (tail) belegt. Diese beiden Element markieren
             den Anfang und das Ende der Liste. Sie enthalten keine Daten
             und werden nie geloescht. Neue Elemente werden deshalb immer
             zwischen zwei bereits vorhandenen Listenelementen eingefuegt,
             Sonderfaelle gibt es nicht.

  Parameter: list - die Liste

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE Init(VAR list:ListType) =
  BEGIN
    list.head := NEW (REF Element);
    list.tail := NEW (REF Element);
    list.head^.next := list.tail;
    list.head^.prev := NIL;
    list.head^.adr  := HeadElement;
    list.tail^.prev := list.head;
    list.tail^.next := NIL;
    list.tail^.adr  := TailElement;
  END Init;

(*---------------------------------------------------------------------------
  Name:      Insert

  Aufgabe:   fuegt eine neue Adresse in die Liste ein. Die Liste wird solange
             durchlaufen, wie der Namen des aktuellen Listenelements in der
             alphabetischen Ordnung vor dem Namen des einzufuegenden Elements
             kommt. Wenn diese Bedingung nicht mehr gilt, gibt es zwei Faelle:
             Entweder die beiden Namen sind gleich, dann soll laut
             Aufgabenstelleung der alten Eintrag einfach durch den neuen
             ueberschrieben werden, oder die beiden Namen sind nicht
             gleich, dann wird ein neues Element in die Liste eingefuegt.

  Parameter: newAdr - die neue Adresse.

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE Insert(VAR list:ListType; newAdr:Address) =
  VAR
    newPointer, helpPointer : REF Element;

  BEGIN
    helpPointer:=list.head;

   (* Suche die Stelle zum Einfuegen in die Liste. *)

    WHILE Text.Compare (helpPointer^.next^.adr.name, newAdr.name) <= 0 AND
          helpPointer^.next # list.tail DO

      helpPointer:=helpPointer^.next;
    END;

    (* Pruefe ob die beiden Namen gleich sind *)
    IF Text.Compare(helpPointer^.adr.name, newAdr.name) = 0 THEN

      helpPointer^.adr := newAdr;   (* alte Adresse wird ersetzt *)
    ELSE
      newPointer := NEW (REF Element);  (* neues Element anlegen und einfuegen *)
      newPointer^.adr := newAdr;
      newPointer^.next := helpPointer^.next;
      newPointer^.prev := helpPointer;
      helpPointer^.next^.prev := newPointer;
      helpPointer^.next := newPointer;
    END;
  END Insert;
(*---------------------------------------------------------------------------
  Name:      Find

  Aufgabe:   Die Prozedur sucht anhand eines Namens nach einem Listeneintrag.
             Die Liste wird solange durchlaufen, bis entweder der gesuchte
             Namen gefunden wurde, oder der aktuelle Namen alphabetisch
             groesser ist, als der gesuchte, oder das Ende der Liste erreicht
             wurde.

  Parameter: list - die Liste
             name - der gesuchte Namen

  Rueckgabe: ein Zeiger auf das gesuchte Element. Falls das Element
             nicht gefunden wurde, wird NIL zurueckgegeben.
---------------------------------------------------------------------------*)
```

```
PROCEDURE Find (list: ListType; name:TEXT): REF Element=
VAR  helpPointer : REF Element;
BEGIN
  helpPointer := list.head^.next;

  (* Suche den Namen *)
  WHILE ((Text.Compare (helpPointer^.adr.name, name) < 0) AND
    (helpPointer # list.tail)) DO
    helpPointer:= helpPointer^.next;
  END;

  (* Wenn Listenende erreicht -> NIL *)
  IF helpPointer = list.tail THEN RETURN NIL;
  ELSE

    (* Wurde der Namen gefunden? *)
    IF Text.Compare(helpPointer^.adr.name, name) = 0 THEN

      RETURN helpPointer; (* Zeiger auf das gesuchte Element *)
    ELSE
      RETURN NIL;         (* sonst NIL zurueckgeben *)
    END;
  END;
END Find;
(*---------------------------------------------------------------------------
  Name:      Delete

  Aufgabe:   Die Prozedur loescht ein Element aus der Liste indem sie
             es aus der Listenstruktur auskettet.

  Parameter: pointer - ein Zeiger auf das Element

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE Delete (pointer: REF Element) =
  BEGIN
    IF pointer # NIL THEN
        pointer^.prev^.next := pointer^.next;
        pointer^.next^.prev := pointer^.prev;
        pointer := NIL;
    END;
  END Delete;
(*---------------------------------------------------------------------------
  Name:      PrintList

  Aufgabe:   gibt die gesamte Liste aus

  Parameter: list - die Liste

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE PrintList(list:ListType) =
VAR helpPointer: REF Element;
    helpText : TEXT;
  BEGIN
    helpPointer := list.head;
    WHILE helpPointer # NIL DO
      WITH h = helpPointer^.adr DO
        helpText := h.name & ", " & h.street & " " & h.num & ", "
                    & h.post & " "& h.city;
        SIO.PutText(helpText);
        SIO.Nl();
      END;
      helpPointer:= helpPointer^.next;
    END;
  END PrintList;
(*---------------------------------------------------------------------------
  Name:      InpuAddress

  Aufgabe:   liest eine Adresse vom Benutzer ein und fuegt sie in
             die Liste ein.

  Parameter: list - die Liste

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE InputAddress(list:ListType) =
VAR newAdr: Address;
  BEGIN
    SIO.PutText ("Name        : ");
    newAdr.name := SIO.GetLine();
    SIO.PutText ("Strasse     : ");
    newAdr.street := SIO.GetLine();
    SIO.PutText ("Hausnummer  : ");
    newAdr.num := SIO.GetLine();
    SIO.PutText ("Postleitzahl: ");
    newAdr.post := SIO.GetLine();
    SIO.PutText ("Ort         : ");
    newAdr.city := SIO.GetLine();

    Insert(list,newAdr);
  END InputAddress;
```

```
(*-------------------------------------------------------------------------------
  Name:       SeachAddress

  Aufgabe:    liest einen Namen vom Benutzer ein und ruft mit diesem Namen
              der Prozedur Find auf um danach zu suchen.

  Parameter: list - die Liste

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE SearchAddress(list:ListType) =
 VAR name:TEXT;
     pointer:REF Element;
 BEGIN
   SIO.PutText("Nach welchem Namen soll gesucht werden: ");
   name:=SIO.GetLine();
   pointer := Find(list,name);
   IF pointer = NIL THEN
     SIO.PutText ("\nDieser Name ist nicht in der Liste!\n")
   ELSE
     SIO.PutText ("Die Adresse lautet: \n\n");
     WITH h = pointer^.adr DO
       SIO.PutText (h.name); SIO.Nl();
       SIO.PutText (h.street & " " & h.num);  SIO.Nl();
       SIO.PutText (h.post   & " " & h.city); SIO.Nl();
       SIO.Nl();
     END;
   END;
 END SearchAddress;

(*-------------------------------------------------------------------------------
  Name:       RemoveName

  Aufgabe:    liest einen Namen vom Benutzer ein und loescht den Eintrag
              mit diesem Namen falls vorhanden.

  Parameter: list - die Liste

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE RemoveName(list:ListType) =
 VAR name:TEXT;
     pointer:REF Element;
 BEGIN
   SIO.PutText("Welcher Namen soll geloescht werden: ");
   name:=SIO.GetLine();
   pointer := Find(list,name);
   IF pointer = NIL THEN
     SIO.PutText ("\nDieser Name ist nicht in der Liste!\n")
   ELSE
     Delete(pointer);
     SIO.PutText ("\nDer Eintrag wurde geloescht!\n")
   END;
 END RemoveName;
(*-------------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------------*)
BEGIN
  Init(List);

  REPEAT
    SIO.PutText("\n----------------------------------\n");
    SIO.PutText("      M E N U E         \n");
    SIO.PutText("                        \n");
    SIO.PutText(" 1 - Neue Adresse eingeben   \n");
    SIO.PutText(" 2 - Anzeigen der Liste      \n");
    SIO.PutText(" 3 - Suchen nach einem Namen \n");
    SIO.PutText(" 4 - Loeschen eines Elements  \n");
    SIO.PutText(" 5 - Beenden des Programms   \n");
    SIO.PutText("\n\n");
    SIO.PutText("Bitte waehlen Sie eine Funktion: ");
    input:=SIO.GetLine() & " ";
    SIO.Nl();
    SIO.PutText("\n----------------------------------\n");
    SIO.Nl();

    CASE Text.GetChar(input,0) OF
       | '1' => InputAddress(List);
       | '2' => PrintList(List);
       | '3' => SearchAddress (List);
       | '4' => RemoveName (List);
    ELSE END;

  UNTIL Text.GetChar(input,0) ='5';
END DoubleList.
```

Aufgabe 7.3

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 7.3 (Zusatzaufgabe)

  Modul:   Pack
  Datei:   Pack.m3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
MODULE Pack EXPORTS Main;

IMPORT SIO,Fmt,Wr,Stdio;

CONST
  MAXWEIGHT = 1000;
  NUMOFPACKETS = 28;

TYPE
  PacketNumbers = [1..NUMOFPACKETS];
  PackNumSet    = SET OF PacketNumbers;
  Packet = RECORD
    Value, Weight: CARDINAL;
  END;

  AllPacketsT = ARRAY PacketNumbers OF Packet;

VAR

  MaxVal: CARDINAL;
  MaxValLoad: PackNumSet;

  AllPackets := AllPacketsT{
  Packet{100,299},  Packet{120,150},
  Packet{200,170},  Packet{80,180},
  Packet{320,300},  Packet{65,254},
  Packet{120,275},  Packet{55,175},
  Packet{20,140},   Packet{65,190},
  Packet{130,180},  Packet{120,240},
  Packet{120,250},  Packet{200,500},
  Packet{89,200},   Packet{150,260},
  Packet{230,405},  Packet{120,1900},
  Packet{110,210},  Packet{202,405},
  Packet{112,266},  Packet{150,190},
  Packet{290,220},  Packet{185,320},
  Packet{140,310},  Packet{190,280},
  Packet{60,210},   Packet{202,202}
  };

(*-----------------------------------------------------------------------------
  Name:      LoadNextPacket

  Aufgabe:   Testet fuer das naechste Paket alle zuerst Kombinationen, in denen
             das Paket zur Ladung gehoert und danach alle Kombinationen, in
             denen das Paket nicht zur Ladung gehoert.

  Parameter: nextToLoad    - das Paket, das als naechstes getestet werden soll
             loadedPackets - die Pakete, die bereits aufgeladen sind
             loadedWeight  - das Gewicht der bereits geladenen Pakete
             loadedValue   - der Wert der bereits geladenen Pakete
             maxValue      - der Wert der wertvollsten Ladung, die bisher
                             gefunden wurde
             maxValueLoad  - die Pakete, aus denen die bisher wertvollste
                             Ladung besteht.

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE LoadNextPacket (nextToLoad: CARDINAL;
                          loadedPackets: PackNumSet;
                          loadedWeight:  CARDINAL;
                          loadedValue: CARDINAL;
                          VAR maxValue: CARDINAL;
                          VAR maxValueLoad: PackNumSet) =

  BEGIN

    IF loadedWeight <= MAXWEIGHT THEN

      IF loadedValue >= maxValue THEN
        maxValue := loadedValue;
        maxValueLoad := loadedPackets;
      END;

      PrintState (maxValue, loadedWeight, loadedPackets);
```

```
      IF nextToLoad <= NUMOFPACKETS THEN

        LoadNextPacket (nextToLoad+1,
                        loadedPackets + PackNumSet{nextToLoad},
                        loadedWeight + AllPackets[nextToLoad].Weight,
                        loadedValue + AllPackets[nextToLoad].Value,
                        maxValue,
                        maxValueLoad);

        LoadNextPacket (nextToLoad+1,
                        loadedPackets,
                        loadedWeight,
                        loadedValue,
                        maxValue,
                        maxValueLoad);

      END;
    END;
  END LoadNextPacket;

(*---------------------------------------------------------------------------
  Name:      PrintState

  Aufgabe:   Gibt den Zwischenzustand der Berechnung aus. Mit dieser Funktion
             kann man nachvollziehen, wie das Programm arbeitet. Es werden
             ausgegeben:
             - maximaler Wert bisher,
             - aktuelles Gewicht
             - die aktuelle Ladung des Schlittens

  Parameter: max     - maximaler Wert bisher
             weight  - aktuelles Gewicht
             packets - die Ladung des Schlittens

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE PrintState(max,weight: CARDINAL; packets: PackNumSet) =
VAR outString:TEXT;
  BEGIN
    outString := Fmt.Int(max) &" /" &  Fmt.Int(weight) &" :";
    FOR i:= FIRST(PacketNumbers) TO LAST (PacketNumbers) DO
      IF i IN packets THEN
        IF i < 10 THEN outString := outString & " "; END;
        outString := outString & Fmt.Int(i) & " ";
      ELSE
        outString := outString & "   ";
      END;
    END;
    Wr.PutText(Stdio.stdout,outString);
    SIO.Nl();
  END PrintState;

(*---------------------------------------------------------------------------
  Name:      PrintResult

  Aufgabe:   Gibt das Ergebnis der Berechnung aus

  Parameter: val     - der Wert der wervollsten Ladung
             packets - die Packete, aus denen die Ladung besteht

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE PrintResult(val: CARDINAL; packets: PackNumSet) =
  BEGIN
    SIO.PutText ("Liste der Packete :");SIO.Nl();
    SIO.PutText ("--------------------");SIO.Nl();

    FOR i:= FIRST(PacketNumbers) TO LAST (PacketNumbers) DO
      IF i IN packets THEN
        SIO.PutText ("(" & Fmt.Int(AllPackets[i].Value) & "," &
          Fmt.Int(AllPackets[i].Weight) & ")");
        SIO.Nl();
      END;
    END;
    SIO.Nl();
    SIO.PutText("Wert: " & Fmt.Int(val) & " DM");
    SIO.Nl();
  END PrintResult;
(*---------------------------------------------------------------------------
  Hauptprogramm
---------------------------------------------------------------------------*)
BEGIN
  MaxVal:=0;
  MaxValLoad:=PackNumSet{};
  LoadNextPacket(FIRST(PacketNumbers),PackNumSet{},0,0,MaxVal,MaxValLoad);
  PrintResult(MaxVal,MaxValLoad);
END Pack.
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **8. Übungsblatt**

Musterlösung

Aufgabe 8.1

Aufgabe 8.2

a)

```
(*-------------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 8.2

  Modul:   MSort
  Datei:   MSort.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------------*)
MODULE MSort EXPORTS Main;

IMPORT SIO;

TYPE
  ListPointer = REF ListElement;   (* Zeigertyp auf Listenelemente *)

  (* Datentyp eines Listenelements *)

  ListElement = RECORD
    next : REF ListElement;  (* Zeiger auf das naechste Element der Liste *)
    data : CARDINAL;         (* gespeichertes Datum *)
  END;

  (* Eine Liste ist eine Datenstruktur, die einen Zeiger auf das erste
     und letzte Element enthaelt und zusaetzlich die Anzahl der Elemente
     speichert. Dadurch wird es einfacher, die Mitte der Liste
     herauszufinden. *)

  ListType = RECORD
    head, tail: ListPointer;
    length    : CARDINAL;
  END;

VAR

  List: ListType; (* Die Liste, die wir im Hauptprogramm mit den
                     Werten aus der Aufgabenstellung fuellen *)

(*-------------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Initialisierung einer Liste. Wenn eine Liste bereits Elemente
             enthaelt, so werden diese durch Aufruf dieser Prozedur geloescht.

  Parameter: list - die Liste

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE Init (VAR list:ListType) =
  BEGIN
    list.head   := NIL;  (* Setze head und tail auf NIL *)
    list.tail   := NIL;
    list.length := 0;    (* Setze die Anzahl der Listenelemente auf 0 *)
  END Init;
```

```
(*-------------------------------------------------------------------------------
  Name:       Append

  Aufgabe:    Die Prozedur haengt an eine Liste ein neues Element an

  Parameter: list - die Liste
             data - der Wert, der in dem neuen Element abgelegt wird.

  Rueckgabe: keine
-------------------------------------------------------------------------------*)

PROCEDURE Append (VAR list:ListType; data:CARDINAL) =
  VAR ptr : ListPointer;
  BEGIN

    (* Unterscheide die beiden Faelle: A) Liste ist leer und B) mindestens
       ein Element ist schon in der Liste *)


    IF list.head = NIL THEN            (* Fall A) *)
      list.head := NEW (ListPointer);
      list.head^.data := data;
      list.head^.next := NIL;
      list.tail := list.head;
    ELSE
      ptr := NEW (ListPointer);        (* Fall B) *)
      ptr^.data := data;
      ptr^.next := NIL;
      list.tail^.next := ptr;
      list.tail := ptr;
    END;
    list.length := list.length+1;
  END Append;

(*-------------------------------------------------------------------------------
  Name:       PrintList

  Aufgabe:    Gibt den Inhalt der Liste auf dem Bildschirm aus.

  Parameter: list - die Liste

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE PrintList (list: ListType) =
  VAR  ptr : ListPointer;
  BEGIN
    ptr := list.head;
    IF list.head= NIL THEN SIO.PutText("Liste ist leer!");
    ELSE
      REPEAT
        SIO.PutInt (ptr^.data);
        SIO.PutText(",");
        ptr := ptr^.next;
      UNTIL ptr = NIL;
    END;

    SIO.Nl();
  END PrintList;

(*-------------------------------------------------------------------------------
  Name:       Mergesort

  Aufgabe:    Die Prozedur realisiert das Sortieren durch Mischen (Mergesort).
              Aus der zu sortierenden Liste, die der Prozedur als Parameter
              uebergeben wird, werden zwei neue Listen aufgebaut, part1 und part2.
              part1 enthaelt alle Werte bis zur Haelfte der zu sortierenden Liste,
              part2 enthaelt alle Werte ab der Haelfte.
              Fuer jede der beiden Teile wird Mergesort rekursiv aufgerufen.
              Die urspruengliche Liste wird danach geloescht und durch
              Zusammenfuegen der beiden sortierten Teillisten wieder aufgebaut.

              Um die Arbeit der Prozedur zu veranschaulichen, gibt sie zu
              Beginn aus, welche Liste sie zum Sortieren erhalten hat und am
              Ende, wie die sortierte Liste aussieht. Die Ausgaben sind
              entsprechend der Rekursionstiefe eingerueckt. Dazu erhaelt die
              Prozedur eine zusaetzlichen Parameter, der die Rekursionstiefe
              mitzaehlt.

  Parameter: list  - die Liste
             level - die aktuelle Rekursionstiefe

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
```

```
PROCEDURE Mergesort(VAR list:ListType; level:CARDINAL) =
  VAR
    part1,part2:ListType;
    ptr : ListPointer;
  BEGIN
    FOR i:=1 TO level DO SIO.PutText(" "); END;
    SIO.PutText("Mergesort:           ");PrintList (list);

    IF list.length > 1 THEN (* Liste hat mehr als ein Element *)

      Init (part1);  (* Teillisten initialisieren *)
      Init (part2);

      (* Fuelle die beiden Teillisten mit dem Inhalt der Liste. *)
      (* Die Grenze der beiden Teile ist bei der Haelfte der Liste. *)

      ptr := list.head;

      (* Alle Werte bis zur Haelfte *)
      FOR i:= 1 TO (list.length DIV 2) DO
        Append(part1,ptr^.data);
        ptr := ptr^.next;
      END;

      (* Alle Werte ab der Haelfte *)
      FOR i:= list.length DIV 2 TO list.length-1 DO
        Append(part2,ptr^.data);
        ptr := ptr^.next;
      END;

      (* Sortiere die beiden Teile durch den rekursiven Aufruf
         von Mergesort *)

      Mergesort(part1, level+1);
      Mergesort(part2, level+1);

      Init (list); (* Loesche die urspruengliche Liste *)

      (* Solange noch in beiden sortierten Teillisten Elemente
         sind: *)

      WHILE part1.head # NIL AND part2.head # NIL DO

        (* Vergleich das vorderste Element in Teil 1 mit dem
           vordersten Element von Teil 2 und haenge das kleinere
           von beiden an die neue Liste an. Gehe danach in der
           Liste, aus der das vorderste Element entnommen wurde,
           ein Element weiter.*)

        IF (part1.head^.data < part2.head^.data) THEN
          Append(list, part1.head^.data);
          part1.head := part1.head^.next;

        ELSE
          Append(list, part2.head^.data);
          part2.head := part2.head^.next;
        END;
      END;

      (* Wenn nur noch in Teil 2 Elemente stehen, haenge sie
         alle an die neue Liste an. *)

      WHILE part2.head # NIL DO
        Append(list, part2.head^.data);
        part2.head := part2.head^.next;
      END;

      (* Wenn nur noch in Teil 1 Elemente stehen, haenge sie
         alle an die neue Liste an. *)

      WHILE  part1.head # NIL DO
        Append(list, part1.head^.data);
        part1.head := part1.head^.next;
      END;
    END;

    (* Gib das Ergebnis der Sortierung auf dem Bildschirm aus. *)

    FOR i:=1 TO level DO SIO.PutText(" "); END;
    SIO.PutText("Sortiert :           ");PrintList (list);
  END Mergesort;
(*---------------------------------------------------------------------------
  Hauptprogramm
 --------------------------------------------------------------------------- *)
BEGIN
  Init (List);        (* Initialisiere die Liste *)
  Append (List, 15);  (* Fuelle die Liste mit den angegebenen Werten *)
  Append (List, 22);
  Append (List, 7);
  Append (List, 14);
  Append (List, 48);
  Append (List, 25);
  Append (List, 20);
  Append (List, 29);
```

```
  Append (List, 9);
  Append (List, 18);
  Append (List, 38);
  Append (List, 51);
  Append (List, 27);
  Append (List, 11);
  Append (List, 31);

  Mergesort (List,0);  (* Sortiere die Liste *)
END MSort.
```

b) Das unter a) angegebene Programm erzeugt die folgenden Ausgaben, anhand derer man die Zwischenzustände beim Sortieren sehen kann:

```
Mergesort:          15,22,7,14,48,25,20,29,9,18,38,51,27,11,31,
 Mergesort:          15,22,7,14,48,25,20,
  Mergesort:          15,22,7,
   Mergesort:          15,
   Sortiert :          15,
   Mergesort:          22,7,
    Mergesort:          22,
    Sortiert :          22,
    Mergesort:          7,
    Sortiert :          7,
   Sortiert :          7,22,
  Sortiert :          7,15,22,
  Mergesort:          14,48,25,20,
   Mergesort:          14,48,
    Mergesort:          14,
    Sortiert :          14,
    Mergesort:          48,
    Sortiert :          48,
   Sortiert :          14,48,
   Mergesort:          25,20,
    Mergesort:          25,
    Sortiert :          25,
    Mergesort:          20,
    Sortiert :          20,
   Sortiert :          20,25,
  Sortiert :          14,20,25,48,
 Sortiert :          7,14,15,20,22,25,48,
 Mergesort:          29,9,18,38,51,27,11,31,
  Mergesort:          29,9,18,38,
   Mergesort:          29,9,
    Mergesort:          29,
    Sortiert :          29,
    Mergesort:          9,
    Sortiert :          9,
   Sortiert :          9,29,
   Mergesort:          18,38,
    Mergesort:          18,
    Sortiert :          18,
    Mergesort:          38,
    Sortiert :          38,
   Sortiert :          18,38,
  Sortiert :          9,18,29,38,
  Mergesort:          51,27,11,31,
   Mergesort:          51,27,
    Mergesort:          51,
    Sortiert :          51,
    Mergesort:          27,
    Sortiert :          27,
   Sortiert :          27,51,
   Mergesort:          11,31,
    Mergesort:          11,
    Sortiert :          11,
    Mergesort:          31,
    Sortiert :          31,
   Sortiert :          11,31,
  Sortiert :          11,27,31,51,
 Sortiert :          9,11,18,27,29,31,38,51,
Sortiert :          7,9,11,14,15,18,20,22,25,27,29,31,38,48,51,
```

Aufgabe 8.3
In der Musterlösung wird Bezug genommen auf die Beweisregeln für Zuweisungen, Sequenzen und `WHILE`-Schleifen, die auf dem Zusatzblatt zur Hörsaalübung am 7.12. aufgeführt sind. Dieses Blatt kann von der Webseite der Vorlesung heruntergeladen werden.
Laut Aufgabenstellung ist das Kriterium für die Korrektheit des Algorithmus gegeben als

$$(x = z * y + r) \wedge (0 \leq r < y) \qquad (1)$$

Diese Aussage soll nach Terminierung des Algorithmus gelten, wenn als Vorbedingung $(x \geq 0) \wedge (y > 0)$ gilt.

Da mit dem Ende der `WHILE`-Schleife auch das Programm endet, muß man zeigen, daß (1) nach Ende der `WHILE`-Schleife gilt. Zunächst sucht man eine passende Invariante *P* der `WHILE`-Schleife, mit deren Hilfe man (1) beweisen kann (siehe Beweisregel für `WHILE`-Schleifen). Die Aussage (1) ist selbst keine Invariante, weil während des Ablaufs der Schleife die Schleifenbedingung $B \equiv_{def} r \geq y$ gilt. Nach Ende der Schleife gilt $\neg B \equiv (r < y)$. Die Aussage (1) läßt sich jedoch umformen zu

$$(x = z * y + r) \wedge (r \geq 0) \wedge (r < y) \qquad (2)$$

oder auch

$$(x = z * y + r) \wedge (r \geq 0) \wedge \neg B \qquad (3)$$

Der vordere Teil von (3) ist die gesuchte Schleifeninvariante *P*.

$$P \equiv_{def} (x = z * y + r) \wedge (r \geq 0) \qquad (4)$$

Um zu beweisen, daß *P* tatsächlich eine Invariante ist, muß man a) die Gültigkeit von *P* vor der `WHILE`-Schleife zeigen und b) zeigen, daß *P* nach einmaligem Durchlaufen der Schleifenrumpfs immer noch gilt, wenn es vorher gültig war.
Wir beginnen, indem wir b) beweisen, der Beweis von a) folgt anschließend:
Wenn *P* nach dem Schleifenrumpf gelten soll, heißt das, daß *P* eine Nachbedingung der Anweisung z:=z+1 sein muß:

$$\mathrm{z} := \mathrm{z} + 1\{(x = z * y + r) \wedge (r \geq 0)\}$$

Durch Anwendung der Zuweisungsregel ermittelt man die notwendige Vorbedingung:

$$\{(x = (z+1) * y + r) \wedge (r \geq 0)\}\mathrm{z} := \mathrm{z} + 1\{(x = z * y + r) \wedge (r \geq 0)\} \qquad (5)$$

Damit diese Vorbedingung gilt, muß sie wiederum Nachbedingung der vorherigen Anweisung sein:

$$\mathrm{r} := \mathrm{r} - \mathrm{y}\{(x = (z+1) * y + r) \wedge (r \geq 0)\}$$

Nochmaliges Anwenden der Zuweisungsregel ergibt als Vorbedingung:

$$\{(x = (z+1) * y + (r - y) \wedge ((r - y) \geq 0)\}\mathrm{r} := \mathrm{r} - \mathrm{y}\{(x = (z+1) * y + r) \wedge (r \geq 0)\}$$

Durch Vereinfachung der Vorbedingung erhält man:

$$\{(x = z * y + r) \wedge (r \geq 0) \wedge (r \geq y)\}\mathrm{r} := \mathrm{r} - \mathrm{y}\{(x = (z+1) * y + r) \wedge (r \geq 0)\} \qquad (6)$$

Weil die Vorbedingung von (5) mit der Nachbedingung von (6) übereinstimmt, kann man die Squenzregel anwenden und erhält:

$$\{(x = z * y + r) \wedge (r \geq 0) \wedge (r \geq y)\}$$
$$\text{r} := \text{r} - \text{y};$$
$$\text{z} := \text{z} + 1;$$
$$\{(x = z + * y + r) \wedge (r \geq 0)\}$$

Mit Hilfe von *P* und *B* ausgedrückt:

$$\{P \wedge B\}\ \text{r} := \text{r} - \text{y}; \text{z} := \text{z} + 1; \{P\}$$

Damit ist gezeigt, daß *P* eine Invariante der `WHILE`-Schleife ist und mit Hilfe der `WHILE`-Regel kann man folgern, daß wenn *P* vor der `WHILE`-Schleife gilt, hinterher *P* und $\neg B$ gelten. Es bleibt noch b) zu zeigen, daß *P* vor der `WHILE`-Schleife gilt. Durch Verwendung der Zuweisungsregel erhält man:

$$\{(x = 0 * y + x) \wedge (x \geq 0)\}\text{z} := 0\{(x = z * y + x) \wedge (x \geq 0)\} \quad (7)$$
$$\{(x = z * y + x) \wedge (x \geq 0)\}\text{r} := \text{x}\{(x = z * y + r) \wedge (r \geq 0)\} \quad (8)$$

Der Beweis, daß aus (7) und (8) mit Hilfe der Sequenzregel folgt, daß die Vorbedingung von (7) für die Sequenz aus beiden Zuweisungen gilt, erfolgt genauso wie für den Rumpf der Schleife und ist deshalb hier nicht angegeben.
Die Vorbedingung von z:=0 kann man vereinfachen zu $(\text{x} = \text{x}) \wedge (\text{x} \geq 0)$, was unmittelbar aus der gegebenen Vorbedingung des gesamten Programmstücks folgt. Damit ist der Beweis vollständig.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja*|*jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **10. Übungsblatt**

Musterlösung

Aufgabe 10.1

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   StackADO
  Datei:   StackADO.i3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
INTERFACE StackADO;

EXCEPTION StackEmpty;
EXCEPTION StackFull;

PROCEDURE Push  (data:INTEGER)       RAISES {StackFull};
PROCEDURE Pop   ()                   RAISES {StackEmpty};
PROCEDURE Item  ()          :INTEGER RAISES {StackEmpty};
PROCEDURE Empty ()          :BOOLEAN;


(* ANMERKUNG: Die Prozedur PrintStack wird in diesem Interface nur
   exportiert, damit man sich im Hauptprogramm den internen Aufbau und
   die Arbeitsweise von StackADO ansehen kann. Normalerweise gehoert
   eine solche Prozedur AUF KEINEN FALL in das Interface, denn der Sinn
   eines ADO ist ja gerade, die interne Implementierung der Datenstruktur
   vor dem Benutzer des ADO zu verbergen! *)

PROCEDURE PrintStack();

END StackADO.
```

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   StackADO
  Datei:   StackADO.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
MODULE StackADO;

IMPORT SIO,Fmt;

CONST
  MAXHEAP = 20;   (* Anzahl der Element und Index des letzten Elements *)

TYPE
  (* Zuerst definieren wir den Typ eines einzelnen Heap Elements *)

  Element = RECORD
    Free: BOOLEAN;   (* speichert die Information, ob dieses Feld frei ist.
                        Fuer das Funktionieren der Implementierung ist diese
                        Komponente nicht notwendig. Sie vereinfacht aber die
                        Prozedur PrintHeap.*)
    Data: INTEGER;   (* die Nutzdaten des Heap Elements *)
    Next: CARDINAL;  (* der Verweis auf das naechste Element der
                        Freispeicherliste *)
  END;
  (* Ein Heap besteht damit aus zwei Koponenten, naemlich dem Index des
     ersten freien Elements und dem Array, das die Elemente enthaelt. *)

  HeapType = RECORD
    First:    CARDINAL;                        (* erster freier Index *)
    Elements: ARRAY [1..MAXHEAP] OF Element;  (* Heap Elemente *)
  END;

VAR
  Heap: HeapType;      (* der Heap, der in diesem Modul verwaltete wird *)
  StackTop:INTEGER;    (* der Index des obersten Elements auf dem Stack *)
(*-------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Die Prozedur initialisiert den Heap. Das Feld First
             wird mit dem Wert 1 initialisiert, d.h. das erste freie Element
             des Heaps soll das Feld des Arrays mit dem Index 1 sein. Die
             Next Felder der Element des Arrays werden mit jeweils mit dem
             Index des naechsten Array Elements initialisiert. Das Feld Next
             des letzten Arraz Elements wird mit dem Wert 0 belegt. Dieser Wert
             markiert das Ende der Freispeicherliste.

  Parameter: heap - die Heap Datenstruktur, die initialisiert wird.

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Init(VAR heap: HeapType) =
  BEGIN
    Heap.First := 1;
    FOR i:=1 TO MAXHEAP-1 DO
      heap.Elements[i].Next := i+1;
      heap.Elements[i].Free := TRUE;
    END;
    heap.Elements[MAXHEAP].Next := 0;
    heap.Elements[MAXHEAP].Free := TRUE;
  END Init;
```

```
(*-------------------------------------------------------------------------
  Name:      New

  Aufgabe:   Die Prozedur liefert den Index eines freien Elements im Heap
             zurueck, falls noch ein freies Element vorhanden ist.
             Das Element, dessen Index zurueckgeliefert wird, wird aus der
             Freispeicherliste herausgenommen.
             Falls kein Element mehr frei ist, wird der Wert 0 zurueckgegeben

  Parameter: heap - der Heap, aus dem ein freies Element entnommen wird

  Rueckgabe: falls noch mindestens ein Element frei ist: der Index eines
                                                         freien Elements.
             falls nicht:                                0
-------------------------------------------------------------------------*)
PROCEDURE New(VAR heap: HeapType):CARDINAL =
  VAR result:CARDINAL;
  BEGIN
    result:=heap.First;
    IF result # 0 THEN
      heap.First:=heap.Elements[result].Next;
      heap.Elements[result].Free := FALSE;
    END;
    RETURN result;
  END New;
(*-------------------------------------------------------------------------
  Name:      Dispose

  Aufgabe:   Die Prozedur gibt ein Element des Heaps, das vorher durch
             die Prozedur New belegt worden ist, wieder frei. Dazu wird
             der uebergebene Index des Elements wieder in die Freispeicherliste
             eingefuegt.

  Parameter: heap  - der Heap, in dem das Element wieder freigegeben werden soll
             index - der Index des Elements

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Dispose(VAR heap:HeapType; index:CARDINAL) =
  BEGIN
    IF index > 0 AND index <= MAXHEAP THEN
      heap.Elements[index].Next := heap.First;
      heap.Elements[index].Free := TRUE;
      heap.First := index;
    END;
  END Dispose;
(*-------------------------------------------------------------------------
  Name:      PrintHeap

  Aufgabe:   Die Prozedur gibt den Inhalt des Heaps auf dem Bildschirm aus.
             Die freien Felder werden durch ein vorangestelltes 'F' markiert.

  Parameter: heap - der Heap, der angezeigt werden soll

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintHeap (VAR heap: HeapType) =
  BEGIN
    SIO.PutText("--------------------------");SIO.Nl();
    SIO.PutText("First: " & Fmt.Int(heap.First));SIO.Nl();
    FOR i:=1 TO MAXHEAP DO
      IF heap.Elements[i].Free THEN
        SIO.PutText("F: ");
      ELSE
        SIO.PutText(" : ");
      END;
```

```
      SIO.PutText(Fmt.Int(i));
      SIO.PutText(" next: " & Fmt.Int(heap.Elements[i].Next));
      SIO.PutText(" data: " & Fmt.Int(heap.Elements[i].Data));
      SIO.Nl();
    END;
    SIO.PutText("-------------------------"); SIO.Nl();
  END PrintHeap;

(*-------------------------------------------------------------------------
  Name:      Push

  Aufgabe:   legt ein neues Element auf dem Stack ab. Dazu wird zuerst durch
             Aufruf der Prozedur New der Index eines freien Heap-Elements
             geholt. Wenn kein Element mehr frei ist, wird die Ausnahme
             'StackFull' erzeugt. Der Index des bisherigen obersten Elements
             des Stacks wird im Feld Next des neuen Elements gespeichert.
             Dadurch ergibt sich die Verkettung des Stacks. Zuletzt wird
             StackTop auf das neu belegte Element gesetzt.

  Parameter: data - der Integerwert, der auf dem Stack abgelegt werden soll.

  Rueckgabe: keine

  Ausnahmen: StackFull - Auf dem Heap, mit dem der Stack intern implementiert
                         wird ist kein Platz mehr frei.
-------------------------------------------------------------------------*)
PROCEDURE Push(data:INTEGER) RAISES {StackFull} =
VAR newIndex:INTEGER;
BEGIN
  newIndex:=New(Heap);                          (* hole neuen Index *)
  IF newIndex = 0 THEN RAISE StackFull;         (* kein freies Feld mehr *)
  ELSE
    Heap.Elements[newIndex].Next := StackTop;   (* verkette mit den Stack *)
    Heap.Elements[newIndex].Data := data;       (* lege Daten ab *)
    StackTop := newIndex;                       (* neues oberstes Element merken *)
  END;
END Push;
(*-------------------------------------------------------------------------
  Name:      Pop

  Aufgabe:   Entfernt das oberste Element vom Stack

  Parameter: keine

  Rueckgabe: keine

  Ausnahmen: StackEmpty - Es konnte kein Element entfernt werden, weil der
                          Stack leer ist.
-------------------------------------------------------------------------*)
PROCEDURE Pop() RAISES {StackEmpty} =
VAR index:INTEGER;
BEGIN
  IF Empty() THEN RAISE StackEmpty;     (* Stack ist leer *)
  ELSE
    index:= StackTop;                   (* Index des obersten Elements merken *)
                                        (* naechstes Element ist neues oberstes *)
    StackTop := Heap.Elements [StackTop].Next;
    Dispose(Heap,index);                (* gespeichertes Element freigeben *)
  END;
END Pop;
```

```
(*-------------------------------------------------------------------------
  Name:      Item

  Aufgabe:   liefert den Inhalt des obersten Elements auf dem Stack zurueck.

  Parameter: keine

  Rueckgabe: Inhalt des obersten Elements

  Ausnahmen: StackEmpty - Das oberste Element konnte nicht gelesen werden,
                          weil der Stack leer ist.
-------------------------------------------------------------------------*)
PROCEDURE Item():INTEGER RAISES {StackEmpty} =
BEGIN
  IF Empty() THEN RAISE StackEmpty;  (* Stack ist leer *)
  ELSE
    RETURN Heap.Elements[StackTop].Data;      (* Inhalt des obersten Elements *)
  END;
END Item;
(*-------------------------------------------------------------------------
  Name:      Empty

  Aufgabe:   Ermittelt, ob der Stack leer ist

  Parameter: keine

  Rueckgabe: TRUE  - Stack ist leer
             FALSE - Stack ist nicht leer
-------------------------------------------------------------------------*)
PROCEDURE Empty():BOOLEAN =
BEGIN
  RETURN StackTop=0;
END Empty;
(*-------------------------------------------------------------------------
  Name:      PrintStack

  Aufgabe:   Gibt den Heap aus, mit dem der Stack implementiert ist

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintStack () =
  BEGIN
    PrintHeap(Heap);
  END PrintStack;
(*-------------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
  Init(Heap);              (* Initialisieren des Heaps *)
  StackTop := 0;           (* Kein Element auf dem Stack abgelegt *)
END StackADO.


(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   StackUser
  Datei:   StackUser.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
```

```
MODULE StackUser EXPORTS Main;

IMPORT StackADO;
IMPORT SIO, Text;

VAR input:TEXT;

BEGIN
  REPEAT
    TRY
      SIO.PutText("\n---------------------------------\n");
      SIO.PutText("      M E N U E         \n");
      SIO.PutText("                        \n");
      SIO.PutText(" 1 - Neues Datum eingeben  \n");
      SIO.PutText(" 2 - Anzeigen des Stacks   \n");
      SIO.PutText(" 3 - Oberstes Datum ansehen \n");
      SIO.PutText(" 4 - Oberstes Datum entfernen  \n");
      SIO.PutText(" 5 - Testen, ob Stack leer ist \n");
      SIO.PutText(" 6 - Beenden des Programms   \n");
      SIO.PutText("\n\n");
      SIO.PutText("Bitte waehlen Sie eine Funktion: ");
      input:=SIO.GetLine() & " ";
      SIO.Nl();
      SIO.PutText("\n---------------------------------\n");
      SIO.Nl();

      TRY
        CASE Text.GetChar(input,0) OF

        | '1' =>
          TRY
            StackADO.Push(SIO.GetInt());
            EVAL SIO.GetLine();
          EXCEPT
            SIO.Error => SIO.PutText("Bitte geben Sie eine Integerzahl ein!\n");
          END;

        | '2' =>
          StackADO.PrintStack();

        | '3' =>
          SIO.PutText("Das oberste Datum des Stacks ist: ");
          SIO.PutInt(StackADO.Item());
          SIO.Nl();

        | '4' =>
          StackADO.Pop();
          SIO.PutText("Das oberste Datum wurde entfernt.\n");

        | '5' =>
          IF StackADO.Empty() THEN
            SIO.PutText("Der Stack ist leer.\n");
          ELSE
            SIO.PutText("Der Stack ist nicht leer.\n");
          END;
        ELSE END;
      EXCEPT
      | StackADO.StackFull => SIO.PutText("*** Fehler: Der Stack ist voll! ***");
      | StackADO.StackEmpty => SIO.PutText ("*** Fehler: Der Stack ist leer! ***");
      END;
    EXCEPT
      SIO.Error  => SIO.PutText("*** Fehler bei der Eingabe***");
    END;
  UNTIL Text.GetChar(input,0) ='6';
END StackUser.
```

Aufgabe 10.2

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.2

  Modul:   StackADO
  Datei:   StackADO.i3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
INTERFACE StackADT;

EXCEPTION StackEmpty;

TYPE T<:REFANY;

PROCEDURE Push  (VAR stack:T; data:INTEGER);
PROCEDURE Pop   (VAR stack:T)                       RAISES {StackEmpty};
PROCEDURE Item  (    stack:T)               :INTEGER RAISES {StackEmpty};
PROCEDURE Empty (    stack:T)               :BOOLEAN;
PROCEDURE Init  (VAR stack:T);

(* ANMERKUNG: Die Prozedur PrintStack wird in diesem Interface nur
   exportiert, damit man sich im Hauptprogramm den internen Aufbau und
   die Arbeitsweise von StackADO ansehen kann. Normalerweise gehoert
   eine solche Prozedur AUF KEINEN FALL in das Interface, denn der Sinn
   eines ADO ist ja gerade, die interne Implementierung der Datenstruktur
   vor dem Benutzer des ADO zu verbergen! *)

PROCEDURE PrintStack(stack:T);

END StackADT.
```

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.2

  Modul:   StackADT
  Datei:   StackADT.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE StackADT;

IMPORT SIO;

REVEAL T = BRANDED REF ListElement;

TYPE
  ListElement = RECORD
    data : INTEGER;
    next : T;
  END;
```

```
(*-------------------------------------------------------------------------
  Name:      Push

  Aufgabe:   legt ein neues Element auf dem Stack ab.

  Parameter: data - der Integerwert, der auf dem Stack abgelegt werden soll.

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Push(VAR stack:T; data:INTEGER) =
VAR ptr:T;
BEGIN
  ptr := NEW (T);
  ptr^.data := data;
  ptr^.next := stack;
  stack := ptr;
END Push;
(*-------------------------------------------------------------------------
  Name:      Pop

  Aufgabe:   Entfernt das oberste Element vom Stack

  Parameter: keine

  Rueckgabe: keine

  Ausnahmen: StackEmpty - Es konnte kein Element entfernt werden, weil der
                          Stack leer ist.
-------------------------------------------------------------------------*)
PROCEDURE Pop(VAR stack:T) RAISES {StackEmpty} =
BEGIN
  IF Empty(stack) THEN RAISE StackEmpty;     (* Stack ist leer *)
  ELSE
    stack:=stack^.next;
  END;
END Pop;
(*-------------------------------------------------------------------------
  Name:      Item

  Aufgabe:   liefert den Inhalt des obersten Elements auf dem Stack zurueck.

  Parameter: keine

  Rueckgabe: Inhalt des obersten Elements

  Ausnahmen: StackEmpty - Das oberste Element konnte nicht gelesen werden,
                          weil der Stack leer ist.
-------------------------------------------------------------------------*)
PROCEDURE Item(stack:T):INTEGER RAISES {StackEmpty} =
BEGIN
  IF Empty(stack) THEN RAISE StackEmpty;  (* Stack ist leer *)
  ELSE
    RETURN stack^.data;      (* Inhalt des obersten Elements *)
  END;
END Item;
```

```
(*-------------------------------------------------------------------------
  Name:      Empty

  Aufgabe:   Ermittelt, ob der Stack leer ist

  Parameter: keine

  Rueckgabe: TRUE  - Stack ist leer
             FALSE - Stack ist nicht leer
-------------------------------------------------------------------------*)
PROCEDURE Empty(stack:T):BOOLEAN =
BEGIN
  RETURN stack=NIL;
END Empty;
(*-------------------------------------------------------------------------
  Name:      PrintStack

  Aufgabe:   Gibt den Heap aus, mit dem der Stack implementiert ist

  Parameter: stack

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintStack (stack:T) =
VAR ptr:T;
BEGIN
  ptr:=stack;
  WHILE ptr # NIL DO
    SIO.PutInt(ptr^.data); SIO.Nl();
    ptr:=ptr^.next;
  END;
  SIO.Nl();
END PrintStack;
(*-------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Initialisiert den Stack indem der Zeiger auf die Liste
             der Stackelemente auf NIL gesetzt wird.

  Parameter: stack

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Init (VAR stack:T) =
  BEGIN
    stack:=NIL;
  END Init;
(*-------------------------------------------------------------------------
Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
END StackADT.



(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.2

  Modul:   StackUser
  Datei:   StackUser.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
```

```
MODULE StackUser EXPORTS Main;

IMPORT StackADT;
IMPORT SIO, Text;

VAR input:TEXT;
    Stack:StackADT.T;

BEGIN
  StackADT.Init(Stack);
  REPEAT
    TRY
      SIO.PutText("\n---------------------------------\n");
      SIO.PutText("      M E N U E         \n");
      SIO.PutText("                        \n");
      SIO.PutText(" 1 - Neues Datum eingeben  \n");
      SIO.PutText(" 2 - Anzeigen des Stacks   \n");
      SIO.PutText(" 3 - Oberstes Datum ansehen \n");
      SIO.PutText(" 4 - Oberstes Datum entfernen  \n");
      SIO.PutText(" 5 - Testen, ob Stack leer ist \n");
      SIO.PutText(" 6 - Beenden des Programms   \n");
      SIO.PutText("\n\n");
      SIO.PutText("Bitte waehlen Sie eine Funktion: ");
      input:=SIO.GetLine() & " ";
      SIO.Nl();
      SIO.PutText("\n---------------------------------\n");
      SIO.Nl();

      TRY
        CASE Text.GetChar(input,0) OF

        | '1' =>
          TRY
            StackADT.Push(Stack,SIO.GetInt());
            EVAL SIO.GetLine();
          EXCEPT
            SIO.Error => SIO.PutText("Bitte geben Sie eine Integerzahl ein!\n");
          END;

        | '2' =>
          StackADT.PrintStack(Stack);

        | '3' =>
          SIO.PutText("Das oberste Datum des Stacks ist: ");
          SIO.PutInt(StackADT.Item(Stack));
          SIO.Nl();

        | '4' =>
          StackADT.Pop(Stack);
          SIO.PutText("Das oberste Datum wurde entfernt.\n");

        | '5' =>
          IF StackADT.Empty(Stack) THEN
            SIO.PutText("Der Stack ist leer.\n");
          ELSE
            SIO.PutText("Der Stack ist nicht leer.\n");
          END;
        ELSE END;
      EXCEPT
      | StackADT.StackEmpty => SIO.PutText ("*** Fehler: Der Stack ist leer! ***");
      END;
    EXCEPT
      SIO.Error  => SIO.PutText("*** Fehler bei der Eingabe***");
    END;
  UNTIL Text.GetChar(input,0) ='6';
END StackUser.
```

Aufgabe 10.3

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.3

  Modul:   Route
  Datei:   Route.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE Route EXPORTS Main;
IMPORT SIO,Fmt;

CONST NUMOFCITIES = 10;

TYPE
  Cities    = {Aachen, Bamberg, Berlin, Bonn, Duesseldorf, Essen, Hamburg, Koeln,
               Muenchen, Stuttgart};
  CitySet   = SET OF Cities;

  DistMatrixLineT = ARRAY [Cities.Aachen..Cities.Stuttgart] OF CARDINAL;
  DistMatrixT     = ARRAY [Cities.Aachen..Cities.Stuttgart] OF DistMatrixLineT;

  Tour = RECORD
    cities:ARRAY[1..NUMOFCITIES] OF Cities;
    next:CARDINAL;
  END;


VAR
  CityNames := ARRAY [Cities.Aachen..Cities.Stuttgart] OF TEXT
      {"Aachen      ",
       "Bamberg     ",
       "Berlin      ",
       "Bonn        ",
       "Duesseldorf ",
       "Essen       ",
       "Hamburg     ",
       "Koeln       ",
       "Muenchen    ",
       "Stuttgart   "};

  Distances := DistMatrixT{
      DistMatrixLineT{0,   0,   0,   0,   0,   0,   0,   0,   0,  0},
      DistMatrixLineT{455, 0,   0,   0,   0,   0,   0,   0,   0,  0},
      DistMatrixLineT{644, 404, 0,   0,   0,   0,   0,   0,   0,  0},
      DistMatrixLineT{94,  378, 633, 0,   0,   0,   0,   0,   0,  0},
      DistMatrixLineT{79,  417, 571, 70,  0,   0,   0,   0,   0,  0},
      DistMatrixLineT{117, 450, 527, 103, 38,  0,   0,   0,   0,  0},
      DistMatrixLineT{486, 517, 285, 454, 407, 369, 0,   0,   0,  0},
      DistMatrixLineT{72,  374, 570, 27,  43,  76,  427, 0,   0,  0},
      DistMatrixLineT{628, 240, 592, 558, 628, 643, 757, 567, 0,  0},
      DistMatrixLineT{480, 244, 624, 362, 403, 436, 690, 360, 218,0}};

  MinTourDistance: CARDINAL;
  MinTour:         Tour;
```

```
(*-------------------------------------------------------------------------
  Name:      GetDistance

  Aufgabe:   ermittelt die Entfernung zwischen zwei Staedten anhand der
             Entfernungsmatrix

  Parameter: from, to  - die beiden Staedte

  Rueckgabe: Entfernung zwischen form und to
-------------------------------------------------------------------------*)
PROCEDURE GetDistance(from,to:Cities):CARDINAL =
  BEGIN
    IF from < to THEN
      RETURN Distances[to,from]
    ELSE
      RETURN Distances[from,to]
    END;
  END GetDistance;
(*-------------------------------------------------------------------------
  Name:      TourDistance

  Aufgabe:   Ermittelt die Laenge einer Rundreise

  Parameter: tour - die Rundreise

  Rueckgabe: Laenge der Tour
-------------------------------------------------------------------------*)
PROCEDURE TourDistance(tour: Tour):CARDINAL =
  VAR dist:CARDINAL;
  BEGIN
    dist:=0;
    IF tour.next > 2 THEN
      FOR i:=2 TO tour.next-1 DO
        dist:=dist+GetDistance(tour.cities[i],tour.cities[i-1]);
      END;
      dist:=dist+GetDistance(tour.cities[1],tour.cities[tour.next-1]);
    END;
    RETURN dist;
  END TourDistance;
(*-------------------------------------------------------------------------
  Name:      AddCity

  Aufgabe:   fuegt zur einer Rundreise ein neue Stadt hinzu

  Parameter: tour - die Rundreise
             city - die Stadt

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE AddCity (VAR tour: Tour; city:Cities) =
  BEGIN
    IF tour.next <= NUMOFCITIES THEN
      tour.cities[tour.next]:=city;
      tour.next := tour.next+1;
    END;
  END AddCity;

(*-------------------------------------------------------------------------
  Name:      InitTour

  Aufgabe:   Initialisiert die Datenstruktur zur Verwaltung einer Rundreise

  Parameter: tour - die Rundreise

  Rueckgabe: keine
-------------------------------------------------------------------------*)
```

```
PROCEDURE InitTour (VAR tour:Tour) =
  BEGIN
    tour.next:=1;
  END InitTour;

(*---------------------------------------------------------------------------
  Name:      TourComplete

  Aufgabe:   Gibt an, ob eine Rundreise vollstaendig ist.

  Parameter: tour - die Rundreise

  Rueckgabe: TRUE genau dann wenn die Rundreise vollstaendig ist.
---------------------------------------------------------------------------*)
PROCEDURE TourComplete (tour:Tour):BOOLEAN =
  BEGIN
    RETURN tour.next > NUMOFCITIES;
  END TourComplete;

(*---------------------------------------------------------------------------
  Name:      PrintTour

  Aufgabe:   Gibt den Verlauf einer Rundreise aus.

  Parameter: minDist - die Laenge der bisher kuerzesten Rundreise
             tour    - die auszugebende Rundreise

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE PrintTour(minDist:CARDINAL; tour: Tour) =
VAR outString:TEXT;
  BEGIN
    outString := Fmt.Int(minDist) &" /" &  Fmt.Int(TourDistance(tour)) &" :";
    FOR i:= 1 TO tour.next-1 DO
      outString := outString & CityNames[tour.cities[i]];
    END;
    SIO.PutText(outString);
    SIO.Nl();
  END PrintTour;

 (*---------------------------------------------------------------------------
  Name:      VisitNextCity

  Aufgabe:   fuegt eine neue Stadt in eine Rundreise ein. Wenn die Rundreise
             nach Einfuegen der Stadt immer noch kuerzer ist, als die bisher
             kuerzestet Rundreise durch alle Staedte, ruft sich die Prozedur
             rekursiv auf, um die naechste Stadt einzufuegen usw. Wenn die
             Rundreise nach Einfuegen der Stadt langer ist, als die bisher
             kuerzeste Rundreise, wird die Rekursion abgebrochen. Wenn die
             Rundreise vollstaendig ist, wird sie ausgegeben. Falls sie kuerzer
             ist, als die bisher kuerzeste Rundreise, wird sie gespeichert.

  Parameter: nextToVisit     - die Stadt, die zur Rundreise hinzugefuegt wird
             tour            - die bisherige Rundreise
             citiesVisited   - die Menge der bereits besuchten Staedte
             minTour         - die bislang kuerzeste vollstaendige Rundreise
             minTourDistance - die Laenge der bislang kuerzesten vollstaendigen
                               Rundreise.

  Rueckgabe: keine
---------------------------------------------------------------------------*)
```

```
PROCEDURE VisitNextCity  (nextToVisit: Cities;
                          tour: Tour;
                          citiesVisited: CitySet;
                          VAR minTour: Tour;
                          VAR minTourDistance: CARDINAL) =

  VAR distance:CARDINAL;

  BEGIN

    (* fuege neue Stadt zur Rundreise hinzu *)

    AddCity(tour,nextToVisit);
    citiesVisited := citiesVisited + CitySet{nextToVisit};
    distance := TourDistance(tour);

    IF distance < minTourDistance THEN

      (* Wenn die Laenge der Rundreise immer noch kleiner ist, als die Laege
         der bisher kuerzesten vollstaendigen Rundreise *)

      IF TourComplete(tour) THEN     (* Wenn die Rundreise vollstaendig ist *)
        minTourDistance := distance;         (* speichere als neue kuerzeste *)
        minTour         := tour;             (* Rundreise *)
        PrintTour(minTourDistance, tour);
      ELSE
        (* Ansonsten: Hinzufuegen der naechsten Stadt durch rekursiven Aufruf *)

        FOR i:= FIRST(Cities) TO LAST(Cities) DO
          IF NOT i IN citiesVisited THEN
            VisitNextCity(i,tour,citiesVisited,minTour,minTourDistance);
          END;
        END
      END;

    END;
  END VisitNextCity;

 (*---------------------------------------------------------------------------
  Hauptprogramm
---------------------------------------------------------------------------*)
BEGIN
  InitTour(MinTour);
  MinTourDistance := LAST(INTEGER);

  (*Die Suche nach der kuerzesten Rundreise wird gestartet, indem man
    eine Stadt als Startort vorgibt und VisitNextCity fuer diese Stadt
    aufruft. Anmerkung: Man kann die erste Stadt willkuerlich waehlen
    (z.B. Aachen), weil es sich um eine RUNDREISE handelt und es somit
    nicht auf den Startort ankommt, ebensowenig wie auf die Richtung.
    (Vergleiche dazu z.B. das Ergebnis das dieses Programm fuer den
    Startort Bamberg liefert.) *)

  VisitNextCity(Cities.Aachen,MinTour,CitySet{},MinTour,MinTourDistance);
  SIO.PutText("\nDie kuerzeste Rundreise ist:\n");
  PrintTour(MinTourDistance, MinTour);

END Route.
```

Die kürzeste Rundreise ist Aachen – Düsseldorf – Essen – Hamburg – Berlin – Bamberg – München – Stuttgart – Bonn – Köln – Aachen mit einer Länge von 2094 km.

Aufgabe 10.4

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.4

  Modul:   Tree
  Datei:   Tree.i3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
INTERFACE Tree;

EXCEPTION NoMoreQuestions;
EXCEPTION CurrentQuestionNotFinal;

PROCEDURE InitTree();
PROCEDURE ResetToFirst();
PROCEDURE GetQuestion(): TEXT RAISES {NoMoreQuestions};
PROCEDURE Answer(answer: BOOLEAN):BOOLEAN;
PROCEDURE HasMoreQuestions():BOOLEAN;
PROCEDURE LearnFirst (name:TEXT);
PROCEDURE LearnNext (name: TEXT; question: TEXT; answer: BOOLEAN)
  RAISES {CurrentQuestionNotFinal};

END Tree.



(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.4

  Modul:   Tree
  Datei:   Tree.m3
  Autor:   Dirk Jaeger

  Das Modul implementiert ein abstraktes Datenobjekt (ADO) 'Ratebaum'.
  Innerhalb eines Ratebaums gibt es zwei Arten von Fragen: Unterscheidungs-
  fragen und konkrete Fragen. Unterscheidungsfragen sind die Knoten des
  Baumes, die noch Nachfolgerknoten haben, also z.B. im Fall des Tierraetsels
  die Frage: "Ist es ein Saeugetier?". Konkrete Fragen fragen nach einem
  bestimmten Tier, wie z.B.: "Ist es ein Elefant?" Eine konkrete Frage hat
  keinen Nachfolger. Die Knoten, in denen die konkreten Fragen gespeichert
  sind, bilden die Blaetter des Baumes. Am Anfang wird der Baum mit
  einer konkreten Frage initialisiert. Bei einer Erweiterung des Baumes werden
  immer eine konkrete und eine Unterscheidungsfrage in den Baum eingefuegt
  (siehe Aufgabenstellung).
---------------------------------------------------------------------------*)
MODULE Tree;

CONST
  FinalQuestion = "Ist es ein(e) ";  (* Der Standard Text fuer eine
                                        konkrete Frage *)

TYPE
  NodePtr = REF TreeNode; (* Ein Zeiger auf einen Knoten des Ratebaums *)
```

```
  (* die Datenstruktur eines Knotens *)

  TreeNode = RECORD
    question:  TEXT;      (* Der Text der Frage *)
    yes     :  NodePtr;   (* Ein Zeiger auf den Knoten, an dem die naechste
                             Frage steht fuer den Fall, dass der Benutzer
                             mit Ja antwortet *)
    no      :  NodePtr;   (* desgleichen fuer die Antwort Nein *)
    parent  :  NodePtr;   (* der uebergeordnete Knoten im Baum *)
  END;

VAR
  Finished    : BOOLEAN;  (* zeigt an, ob bereits richtig geraten wurde *)
  CurrentNode : NodePtr;  (* Der Knoten mit der naechsten Frage *)
  TreeRootPtr : NodePtr;  (* Die Wurzel des Ratebaums *)

(*-------------------------------------------------------------------------
  Name:      InitTree

  Aufgabe:   Die Prozedur initialisiert den Baum indem sie den Zeiger auf
             die Wurzel auf NIL setzt.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE  InitTree() =
  BEGIN
    TreeRootPtr  := NIL;
    Finished     := TRUE;
  END InitTree;

(*-------------------------------------------------------------------------
  Name:      ResetToFirst

  Aufgabe:   Setzt den Zeiger, der auf den Knoten mit der naechsten Frage
             verweist, auf die selbe Adresse wie TreeRootPtr, d.h. die
             naechste Frage ist die oberste Frage im Baum. Finished wird
             genau dann auf TRUE gesetzt, wenn es im Baum noch keine
             Knoten gibt.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE ResetToFirst() =
  BEGIN
    CurrentNode := TreeRootPtr;
    Finished    := CurrentNode=NIL;
  END ResetToFirst;
(*-------------------------------------------------------------------------
  Name:      GetQuestion

  Aufgabe:   Die Prozedur liefert die Frage, die am aktuellen Knoten des Baums
             gespeichert ist, falls noch Fragen uebrig sind, die noch nicht
             gestellt wurden. Falls keine Fragen mehr uebrig sind (Finished ist
             gleich TRUE) wird eine Ausnahme erzeugt.

  Parameter: keine

  Rueckgabe: die aktuelle Frage

  Ausnahmen: Wenn die Prozedur aufgerufen wird wenn keine weitere Frage mehr
             vorhanden ist, wird eine Ausnahme erzeugt NoMoreQuestions
             erzeugt.
-------------------------------------------------------------------------*)
```

```
PROCEDURE GetQuestion(): TEXT RAISES {NoMoreQuestions} =
  BEGIN
    IF NOT Finished THEN
      RETURN CurrentNode.question;
    ELSE
      RAISE NoMoreQuestions;
    END;
  END GetQuestion;

(*-------------------------------------------------------------------------
  Name:      Answer

  Aufgabe:   Durch die Prozedur wird die Antwort des Benutzer verarbeitet.
             Wenn die letzte Frage eine Frage nach einem konkreten Tier war,
             dann gibt es keine weiteren Fragen mehr, d.h. Finished wird
             auf TRUE gesetzt. Wenn es sich um eine Frage zur Unterscheidung
             gehandelt hat, wird in den jeweiligen Teilbaum verzweigt.

  Parameter: answer - die Anwort des Benutzers auf die letzte Frage

  Rueckgabe: TRUE - wenn der Benutzer auf eine Frage nach einem konkreten Tier
                    mit ja geantwortet hat. Dadurch erfaehrt die aufrufende
                    Prozedur, dass der Benutzer das Spiel gewonnen hat.
             FALSE - sonst.
-------------------------------------------------------------------------*)
PROCEDURE Answer(answer: BOOLEAN):BOOLEAN =
  BEGIN
    IF CurrentNode.yes = NIL THEN  (* Wenn der Zeiger yes (oder no) auf
                                      NIL zeigt, gibt es unter diesem Knoten
                                      keine weiteren Knoten.*)
      Finished:=TRUE;              (* Die Fragerunde ist damit zu Ende *)
      RETURN answer;
    ELSE
        (* Wenn es noch weitere Fragen gibt, gehe weiter zur naechsten
           Frage im Ja- bzw. Nein-Zweig des Baumes. *)

      IF answer = TRUE THEN
        CurrentNode := CurrentNode.yes;
      ELSE
        CurrentNode := CurrentNode.no;
      END;
      RETURN FALSE;
    END;
  END Answer;

(*-------------------------------------------------------------------------
  Name:      HasMoreQuestions

  Aufgabe:   Die Prozedur liefert den Wert von Finished zurueck. Durch Aufruf
             der Prozedur laesst sich testen, ob das Spiel zu Ende ist. Das
             Spiel ist dann zu Ende, wenn eine konkrete Frage gestellt und
             darauf geantwortet wurde.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE HasMoreQuestions():BOOLEAN =
  BEGIN
    RETURN NOT Finished;
  END HasMoreQuestions;
```

```
(*-------------------------------------------------------------------------
  Name:      LearnFirst

  Aufgabe:   Die Prozedur fuegt die erste konkrete Frage in den Baum ein.
             Ein evtl. schon vorhandener Inhalt des Baumes wird dabei
             geloescht.

  Parameter: name - der Name, nach dem gefragt werden soll

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE LearnFirst (name: TEXT) =
  BEGIN
    TreeRootPtr           := NEW (NodePtr);
    TreeRootPtr^.yes      := NIL;
    TreeRootPtr^.no       := NIL;
    TreeRootPtr^.parent   := NIL;
    TreeRootPtr^.question := FinalQuestion & name & "?";
  END LearnFirst;

(*-------------------------------------------------------------------------
  Name:      LearnNext

  Aufgabe:   erweitert den Baum mit einer neuen Unterscheidungsfrage und eine
             neuen konkreten Frage.

  Parameter: name     - der Name nach dem in der konkreten Frage gefragt werden
                        soll.
             question - der Text der Unterscheidungsfrage
             answer   - TRUE bedeutet, dass fuer die bereits vorhandene konkrete
                        Frage die Untescheidungsfrage mit Ja beantwortet wird.

  Rueckgabe: keine

  Ausnahmen: Die Erweiterung des Baumes kann nur durchgefuehrt werden, wenn die
             aktuelle Frage eine konkrete Frage ist. Wenn das nicht der
             Fall ist, wird die Ausnahme 'CurrentQuestionNotFinal' erzeugt.
-------------------------------------------------------------------------*)
PROCEDURE LearnNext (name: TEXT; question: TEXT; answer: BOOLEAN)
  RAISES {CurrentQuestionNotFinal}=
  VAR
    newQuestionNode, newFinalNode: NodePtr;

  BEGIN
    IF  NOT Finished THEN
      RAISE CurrentQuestionNotFinal;   (* Ausnahme: keine konkrete Frage *)
    ELSE

      (* belege Speicher fuer neue Unterscheidungsfrage *)

      newQuestionNode           := NEW (NodePtr);
      newQuestionNode^.question := question;
      newQuestionNode^.parent   := CurrentNode^.parent;

      (* belege Speicher fuer neue konkrete Frage *)

      newFinalNode              := NEW (NodePtr);
      newFinalNode^.question    := FinalQuestion & name & "?";
      newFinalNode^.yes         := NIL;
      newFinalNode^.no          := NIL;
      newFinalNode^.parent      := newQuestionNode;

      (* Als naechstes werden, in Abhaengigkeit vom Parameter answer,
         die neue Frage und die bereits vorhandene Frage ueber
         die Zeiger yes und no an die neue Unterscheidungsfrage
         angefuegt. *)
```

```
      IF answer = TRUE THEN
        newQuestionNode^.no  := newFinalNode;
        newQuestionNode^.yes := CurrentNode;
      ELSE
        newQuestionNode^.yes := newFinalNode;
        newQuestionNode^.no  := CurrentNode;
      END;

      (* Nun wird die neue Unterscheidungsfrage dort in den
         Baum eingefuegt, wo bisher die bereits vorhandene konkrete
         Frage stand. Dabei sind zwei Faelle zu unterscheiden: *)

      IF CurrentNode^.parent = NIL THEN   (* Fall 1: konkrete Frage war einziger *)
        TreeRootPtr := newQuestionNode;   (* Knoten -> Unterscheidungsfrage an   *)
                                          (* TreeRoot anhaengen *)
      ELSE
            (* Fall 2: konkrete Frage hing bereits an einer Unterscheidungsfrage
               -> neue Unterscheidungsfrage an diesem Knoten anhaengen *)

        IF CurrentNode^.parent^.yes = CurrentNode THEN
          CurrentNode^.parent^.yes := newQuestionNode;
        ELSE
          CurrentNode^.parent^.no := newQuestionNode;
        END;
      END;

      (* zuletzt wird der Zeiger der bereits vorhandenen konkreten Frage, der
         auf den Vorgaenger im Baum verweist, auf die neue Untescheidungsfrage
         gesetzt. *)

      CurrentNode^.parent := newQuestionNode;
    END;
  END LearnNext;
(*-------------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------------*)
BEGIN
  InitTree();  (* Initialisierung des Moduls *)
END Tree.
```

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.4 (Zusatzaufgabe)

  Modul:   Animal
  Datei:   Animal.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
MODULE Animal EXPORTS Main;

IMPORT Tree;       (* Importiere das Modul Tree, dass das abstrakte Datenobjekt
                      zur Speicherung des Ratebaums implementiert. *)
IMPORT SIO,Text;

VAR Continue : BOOLEAN;
(*-------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Initialisiert den Ratebaum mit dem ersten Tier, das das Programm
             kennen soll.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Init() =
  BEGIN
    Tree.LearnFirst("Elefant");
  END Init;
(*-------------------------------------------------------------------------
  Name:      AnswerToBoolean

  Aufgabe:   Dies ist eine Hilfsprozedur, die eine vom Benutzer angegebene
             Antwort in einen BOOLEAN-Wert umwandelt, mit dem man dann
             im Programm einfacher weiterarbeiten kann.

  Parameter: answer - der String, den der Benutzer eingegeben hat.

  Rueckgabe: TRUE, falls der Benutzer "Ja" oder aehnliches eingegeben hat,
             ansonsten FALSE
-------------------------------------------------------------------------*)
PROCEDURE AnswerToBoolean(answer:TEXT): BOOLEAN =
  BEGIN
    RETURN Text.Equal(answer,"JA") OR
           Text.Equal(answer,"ja") OR
           Text.Equal(answer,"Ja") OR
           Text.Equal(answer,"j") OR
           Text.Equal(answer,"J");
  END AnswerToBoolean;
  (*-------------------------------------------------------------------------
  Name:      Learn

  Aufgabe:   Diese Prozedur implementiert den Dialog, mit dem das Programm
             lernt. Das Programm frage nach dem Namen des Tiers, das es nicht
             geraten hat, und nach einer Frage zur Unterscheidung vom
             zuletzt geratenen Tier. Durch Aufruf der Prozedur LearnNext des
             ADO Tree werden die eingegebenen Informationen in den Ratebaum
             eingefuegt.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
```

```
PROCEDURE Learn() =
  VAR
    name,question : TEXT;
    answer : BOOLEAN;
  BEGIN
    TRY
      SIO.PutText("Du hast gewonnen. Ich weiss nicht weiter. Welches Tier war
es?\n");
      name := SIO.GetLine();
      SIO.PutText("Oh, den kannte ich noch nicht! Sag mir doch bitte eine Frage,\n");
      SIO.PutText("durch die ich die beiden unterscheiden kann.\n");
      question := SIO.GetLine();
      SIO.PutText("Wird diese Frage fuer mein geratenes Tier mit JA beantwortet?\n");
      answer:= AnswerToBoolean(SIO.GetLine());
      Tree.LearnNext(name, question, answer);
      SIO.PutText("Danke! Das naechste Mal bin ich schlauer!\n");
    EXCEPT
      | SIO.Error =>
         SIO.PutText(" *** Fehler bei der Eingabe, Lernphase wird beendet! ***");
      | Tree.CurrentQuestionNotFinal =>
         SIO.PutText(" *** Fehler beim der Baumerweiterung, Lernphase wird beendet!
***");
    END;
  END Learn;

(*-------------------------------------------------------------------------
  Name:      Play

  Aufgabe:   Die Prozedur spielt das Ratespiel mit dem Benutzer. Sie holt sich
             aus dem Ratebaum jeweils die naechste Frage und stellt sie dem
             Benutzer. Die vom Benutzer eingegebenen Antwort wird an den
             Ratebaum weitergegeben und bestimmt dadurch, ob im Ratebaum in den
             Ja- oder in den Nein-Zweig gelaufen werden soll.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Play() =
  BEGIN
    Tree.ResetToFirst();  (* Wir fangen bei der ersten Frage an *)

    TRY
      WHILE Tree.HasMoreQuestions() DO  (* Solange es noch Fragen gibt. *)

        SIO.PutText(Tree.GetQuestion() & "\n"); (* Hole naechste Frage und gib
                                                   sie aus *)

        IF AnswerToBoolean(SIO.GetLine()) THEN  (* Hat der Benutzer Ja eingegeben? *)

          IF Tree.Answer(TRUE) THEN        (* Ist das Tier geraten worden?
                                              ANMKERKUNG: Answer liefert genau
                                              dann TRUE, wenn die letzte gestellte
                                              Frage eine konkrete Frage nach einem
                                              bestimmten Tier war. *)

            SIO.PutText("Ich habe gewonnen!\n");(* -> Programm hat Tier geraten.*)
            RETURN;                            (* Verlasse die Prozedur *)
          END;
        ELSE
          EVAL Tree.Answer(FALSE);  (* Antwort war nein *)
        END;
      END;
      Learn();                     (* Lerne nicht geratenes Tier *)
    EXCEPT
      | SIO.Error =>
```

```
        SIO.PutText(" *** Fehler bei der Eingabe, Spiel wird beendet! ***");
      | Tree.NoMoreQuestions =>
        SIO.PutText ("*** Keine Frage mehr vorhanden, Spiel wird beendet! ***");
    END;
  END Play;
(*---------------------------------------------------------------------------
  Hauptprogramm
  ---------------------------------------------------------------------------*)
BEGIN
  Init();
  REPEAT
    TRY
      SIO.Nl();
      SIO.PutText("Denke Dir ein Tier aus, ich versuche es zu raten!\n\n");
      Play();
      SIO.PutText("Moechtest Du nocheinmal spielen?\n");
      Continue := AnswerToBoolean(SIO.GetLine());
    EXCEPT
      | SIO.Error =>
        SIO.PutText(" *** Fehler bei der Eingabe, Programm wird beendet! ***");
    END;
  UNTIL NOT Continue;
END Animal.
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja*|*jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **11. Übungsblatt**

Musterlösung

Aufgabe 11.1

```
(*-------------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 11.1

  Modul:   AddressStorage
  Datei:   AddressStorage.i3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------------*)
INTERFACE AddressStorage;

TYPE
  AdrStorage <: REFANY;

  Address = RECORD   (* der Type der zu speichernden Adressen *)
    name:   TEXT;
    street: TEXT;
    num:    TEXT;
    post:   TEXT;
    city:   TEXT;
  END;

EXCEPTION AddressNotFound;

PROCEDURE Init    (VAR adrst:AdrStorage) ;
PROCEDURE Insert  (VAR adrst:AdrStorage; newAdr:Address) ;
PROCEDURE Find    (    adrst:AdrStorage; name:TEXT): Address RAISES {AddressNotFound};
PROCEDURE Delete  (VAR adrst:AdrStorage; name:TEXT) RAISES {AddressNotFound};
PROCEDURE PrintAll(    adrst:AdrStorage);

END AddressStorage.


(*-------------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 11.1

  Modul:   AddressStorage
  Datei:   AddressStorage.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------------*)
MODULE AddressStorage;

IMPORT SIO, Text;


(* Diese 'Adressen' werden in das erste bzw. letzte Listenelement
   eingetragen. Diese Elemente markieren die beiden Enden der Liste. *)

CONST HeadElement = Address{"HEAD","---","---","---","---"};
      TailElement = Address{"TAIL","---","---","---","---"};

(* Hier wird definiert, auf was der opake Referenztyp AdrStorage
   tatsaechlich verweist *)

REVEAL AdrStorage = BRANDED REF ListType;

TYPE
  Element = RECORD     (* der Typ eines Listenelements *)
    adr:  Address;
    next: REF Element;
    prev: REF Element;
  END;

  ListType = RECORD    (* der Typ der Liste *)
    head: REF Element; (* Zeiger auf das erste Element *)
    tail: REF Element; (* Zeiger auf das letzte Element *)
  END;
```

```
(*-------------------------------------------------------------------------------
  Name:       Init

  Aufgabe:    Die Prozedur initialisiert den ADT, der durch eine doppelt
              verkettete Liste realisiert wird. Damit man beim Einfuegen
              und Loeschen nicht die Sonderfaelle (leere Liste, einfuegen am
              Ende, einfuegen am Anfang) behandeln muss, wird die Liste
              initial mit einem vordersten Element (head) und einem
              hintersten Element (tail) belegt. Diese beiden Element markieren
              den Anfang und das Ende der Liste. Sie enthalten keine Daten
              und werden nie geloescht. Neue Elemente werden deshalb immer
              zwischen zwei bereits vorhandenen Listenelementen eingefuegt,
              Sonderfaelle gibt es nicht.

  Parameter: adrst - eine Instanz des ADT

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE Init(VAR adrst:AdrStorage) =
  BEGIN
    adrst := NEW(AdrStorage);
    adrst^.head := NEW (REF Element);
    adrst^.tail := NEW (REF Element);
    adrst^.head^.next := adrst^.tail;
    adrst^.head^.prev := NIL;
    adrst^.head^.adr  := HeadElement;
    adrst^.tail^.prev := adrst^.head;
    adrst^.tail^.next := NIL;
    adrst^.tail^.adr  := TailElement;
  END Init;
(*-------------------------------------------------------------------------------
  Name:       Insert

  Aufgabe:    fuegt eine neue Adresse in die Liste ein. Die Liste wird solange
              durchlaufen, wie der Namen des aktuellen Listenelements in der
              alphabetischen Ordnung vor dem Namen des einzufuegenden Elements
              kommt. Wenn diese Bedingung nicht mehr gilt, gibt es zwei Faelle:
              Entweder die beiden Namen sind gleich, dann soll laut
              Aufgabenstelleung der alten Eintrag einfach durch den neuen
              ueberschrieben werden, oder die beiden Namen sind nicht
              gleich, dann wird ein neues Element in die Liste eingefuegt.

  Parameter: adrst  - eine Instanz des ADT
              newAdr - die neue Adresse.

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE Insert(VAR adrst:AdrStorage; newAdr:Address) =
  VAR
    newPointer, helpPointer : REF Element;

  BEGIN
    helpPointer:=adrst^.head;

   (* Suche die Stelle zum Einfuegen in die Liste. *)

    WHILE Text.Compare (helpPointer^.next^.adr.name, newAdr.name) <= 0 AND
          helpPointer^.next # adrst^.tail DO

      helpPointer:=helpPointer^.next;
    END;

    (* Pruefe ob die beiden Namen gleich sind *)
    IF Text.Compare(helpPointer^.adr.name, newAdr.name) = 0 THEN

      helpPointer^.adr := newAdr;   (* alte Adresse wird ersetzt *)
    ELSE
      newPointer := NEW (REF Element);  (* neues Element anlegen und einfuegen *)
      newPointer^.adr := newAdr;
      newPointer^.next := helpPointer^.next;
      newPointer^.prev := helpPointer;
      helpPointer^.next^.prev := newPointer;
      helpPointer^.next := newPointer;
    END;
  END Insert;
```

```
(*-------------------------------------------------------------------------
  Name:       FindElement

  Aufgabe:    Die Prozedur sucht anhand eines Namens nach einem Listeneintrag.
              Die Liste wird solange durchlaufen, bis entweder der gesuchte
              Namen gefunden wurde, oder der aktuelle Namen alphabetisch
              groesser ist, als der gesuchte, oder das Ende der Liste erreicht
              wurde.

  Parameter: adrst - eine Instanz des ADT
              name  - der gesuchte Namen

  Rueckgabe: einen Zeiger auf das gesuchte Element. Falls das gesuchte
              Element nicht in der Liste ist, wird NIL zurueckgegeben
-------------------------------------------------------------------------*)
PROCEDURE FindElement (adrst:AdrStorage; name:TEXT): REF Element=
VAR  helpPointer : REF Element;
BEGIN
  helpPointer := adrst^.head^.next;

  (* Suche den Namen *)
  WHILE ((Text.Compare (helpPointer^.adr.name, name) < 0) AND
    (helpPointer # adrst^.tail)) DO
    helpPointer:= helpPointer^.next;
  END;

  (* Wenn Listenende erreicht -> NIL *)
  IF helpPointer = adrst^.tail THEN RETURN NIL;
  ELSE

    (* Wurde der Namen gefunden? *)
    IF Text.Compare(helpPointer^.adr.name, name) = 0 THEN

      RETURN helpPointer; (* Zeiger auf das gesuchte Element *)
    ELSE
      RETURN NIL;         (* sonst NIL zurueckgeben *)
    END;
  END;
END FindElement;
(*-------------------------------------------------------------------------
  Name:       Find

  Aufgabe:    Die Prozedur sucht nach einem Namen in der Liste und gibt
              die Addresse zu diesem Namen zurueck. Wenn der Name nicht
              in der Liste ist, wird eine Ausnahme erzeugt

  Parameter: adrst - eine Instanz des ADT
              name  - der gesuchte Namen

  Rueckgabe: die Addresse zu dem gesuchten Namen
-------------------------------------------------------------------------*)
PROCEDURE Find(adrst:AdrStorage; name:TEXT): Address RAISES {AddressNotFound} =
VAR pointer: REF Element;
BEGIN
  pointer := FindElement(adrst,name);
  IF pointer # NIL THEN
    RETURN pointer^.adr;
  ELSE
    RAISE AddressNotFound;
  END;
END Find;
(*-------------------------------------------------------------------------
  Name:       Delete

  Aufgabe:    Die Prozedur loescht ein Element aus der Liste indem sie
              es aus der Listenstruktur auskettet.

  Parameter: adrst - eine Instanz des ADT
              name  - der Name des zu loeschenden Elements

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Delete (VAR adrst:AdrStorage; name:TEXT) RAISES {AddressNotFound}=
VAR pointer: REF Element;
  BEGIN
    pointer := FindElement(adrst,name);
    IF pointer # NIL THEN
      pointer^.prev^.next := pointer^.next;
      pointer^.next^.prev := pointer^.prev;
      pointer := NIL;
    ELSE
      RAISE AddressNotFound;
    END;
  END Delete;
```

```
(*-------------------------------------------------------------------------
  Name:      PrintAll

  Aufgabe:   gibt die gesamte Liste aus

  Parameter: adrst - eine Instanz des ADT

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintAll(adrst:AdrStorage) =
VAR helpPointer: REF Element;
    helpText : TEXT;
  BEGIN
    helpPointer := adrst^.head;
    WHILE helpPointer # NIL DO
      WITH h = helpPointer^.adr DO
        helpText := h.name & ", " & h.street & " " & h.num & ", "
                    & h.post & " "& h.city;
        SIO.PutText(helpText);
        SIO.Nl();
      END;
      helpPointer:= helpPointer^.next;
    END;
  END PrintAll;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
END AddressStorage.
```

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   AddressOperations
  Datei:   AddressOperations.i3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
INTERFACE AddressOperations;

IMPORT SIO;

PROCEDURE PrintList() ;
PROCEDURE InputAddress() RAISES {SIO.Error};
PROCEDURE SearchAddress() RAISES {SIO.Error};
PROCEDURE RemoveName() RAISES {SIO.Error};

END AddressOperations.
```

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   AddressOperations
  Datei:   AddressOperations.m3
  Autor:   Dirk Jaeger

  Anmerkung: Das Modul ist als ADO implementiert, d.h. es verkapselt einen
  Datenspeicher fuer Adressen. In der Aufgabenstellung war das nicht
  gefordert. Alternativ waere es auch moeglich, ein reines Funktionsmodul
  zu schreiben, dessen Prozeduren auf dem abstrakten Datentyp AdrStorage
  arbeiten. In diesem Fall bekaemen alle Prozeduren dieses Moduls
  als ersten Parameter zusaetzlich eine Instanz des ADT uebergeben.
-------------------------------------------------------------------------*)
MODULE AddressOperations;

FROM AddressStorage IMPORT Init,Insert,Find,Delete,PrintAll;
FROM AddressStorage IMPORT AdrStorage,Address;
FROM AddressStorage IMPORT AddressNotFound;

IMPORT SIO;

VAR Adrst:AdrStorage;  (* zentraler Datenspeicher des ADO *)
```

```
(*-------------------------------------------------------------------------
  Name:      PrintList

  Aufgabe:   gibt die gesamte Liste aus

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintList() =
  BEGIN
    PrintAll(Adrst);
  END PrintList;
(*-------------------------------------------------------------------------
  Name:      InputAddress

  Aufgabe:   liest eine Adresse vom Benutzer ein und fuegt sie in
             die Liste ein.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE InputAddress() RAISES {SIO.Error}=
VAR newAdr: Address;
  BEGIN
    SIO.PutText ("Name         : ");
    newAdr.name := SIO.GetLine();
    SIO.PutText ("Strasse      : ");
    newAdr.street := SIO.GetLine();
    SIO.PutText ("Hausnummer   : ");
    newAdr.num := SIO.GetLine();
    SIO.PutText ("Postleitzahl: ");
    newAdr.post := SIO.GetLine();
    SIO.PutText ("Ort          : ");
    newAdr.city := SIO.GetLine();
    Insert(Adrst,newAdr);
  END InputAddress;
(*-------------------------------------------------------------------------
  Name:      SearchAddress

  Aufgabe:   liest einen Namen vom Benutzer ein und ruft mit diesem Namen
             der Prozedur Find, auf um danach zu suchen.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE SearchAddress()  RAISES {SIO.Error} =
 VAR name:TEXT;
     adr:Address;
 BEGIN
   SIO.PutText("Nach welchem Namen soll gesucht werden: ");
   name:=SIO.GetLine();
   TRY
     adr := Find(Adrst,name);
     SIO.PutText ("Die Adresse lautet: \n\n");
     SIO.PutText (adr.name); SIO.Nl();
     SIO.PutText (adr.street & " " & adr.num);  SIO.Nl();
     SIO.PutText (adr.post   & " " & adr.city); SIO.Nl();
     SIO.Nl();
   EXCEPT
   | AddressNotFound => SIO.PutText ("\nDieser Name ist nicht in der Liste!\n")
   END;
 END SearchAddress;
(*-------------------------------------------------------------------------
  Name:      RemoveName

  Aufgabe:   liest einen Namen vom Benutzer ein und loescht den Eintrag
             mit diesem Namen falls vorhanden.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE RemoveName()  RAISES {SIO.Error} =
 VAR name:TEXT;
 BEGIN
   SIO.PutText("Welcher Namen soll geloescht werden: ");
   name:=SIO.GetLine();
   TRY
     Delete(Adrst,name);
     SIO.PutText ("\nDer Eintrag wurde geloescht!\n")
   EXCEPT
   | AddressNotFound => SIO.PutText ("\nDieser Name ist nicht in der Liste!\n");
   END;
 END RemoveName;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
  Init(Adrst);  (* Initialisiere den Datenspeicher *)
END AddressOperations.
```

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   AddressMain
  Datei:   AddressMain.m3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
MODULE AddressMain EXPORTS Main;

IMPORT SIO, Text;
FROM AddressOperations IMPORT InputAddress, PrintList, SearchAddress, RemoveName;

VAR input:TEXT;
(*-----------------------------------------------------------------------------
  Hauptprogramm
-----------------------------------------------------------------------------*)
BEGIN
  TRY
    REPEAT
      SIO.PutText("\n---------------------------------\n");
      SIO.PutText("      M E N U E          \n");
      SIO.PutText("                         \n");
      SIO.PutText(" 1 - Neue Adresse eingeben   \n");
      SIO.PutText(" 2 - Anzeigen der Liste      \n");
      SIO.PutText(" 3 - Suchen nach einem Namen \n");
      SIO.PutText(" 4 - Loeschen eines Elements  \n");
      SIO.PutText(" 5 - Beenden des Programms   \n");
      SIO.PutText("\n\n");
      SIO.PutText("Bitte waehlen Sie eine Funktion: ");
      input:=SIO.GetLine() & " ";
      SIO.Nl();
      SIO.PutText("\n---------------------------------\n");
      SIO.Nl();

      CASE Text.GetChar(input,0) OF
      | '1' => InputAddress();
      | '2' => PrintList();
      | '3' => SearchAddress ();
      | '4' => RemoveName ();
      ELSE END;

    UNTIL Text.GetChar(input,0) ='5';
  EXCEPT SIO.Error => SIO.PutText("*** Fehler im Modul SIO ***\n");
  END;
END AddressMain.
```

Aufgabe 11.2

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 11.2

  Modul:   InterfaceGenerator
  Datei:   InterfaceGenerator.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE InterfaceGenerator EXPORTS Main;

IMPORT SIO,IO,Text;
IMPORT Wr, Rd;
IMPORT Thread;

(* Anmerkung zu Ausnahmen: Das Programm prueft nicht, ob eine Modula-3 Datei
   syntaktisch korrekt ist. Wenn eine Prozedur nicht korrekt deklariert wird,
   dann wird sie, und evtl. auch nachfolgende, korrekt deklarierte Prozeduren,
   nicht erkannt. Als gesonderte Fehlerfaelle werden lediglich betrachtet,
   dass das Ende des Texte erreicht wird, waehrend das Programm gerade eine
   Textkonstante oder einen Kommentar ueberspringt. *)

EXCEPTION EndOfFileInComment;      (* Textende liegt in einem Kommentar *)
EXCEPTION EndOfTextInTextLiteral;  (* Textende lieft in einer Textkonstanten *)
EXCEPTION EndOfText;               (* Programm ist am Textende angelangt *)

TYPE ModuleText = RECORD   (* Datentyp, der einen Text gemeinsam mit einem *)
  txt: TEXT;               (* Index speichert, der in den Text verweist. *)
  pos: CARDINAL;
END;

TYPE ProcedureHead = RECORD (* Datentyp, der den gesamten Text eines *)
  txt:  TEXT;               (* Prozedurkopfs gemeinsam mit dem Namen der *)
  name: TEXT;               (* Prozedur speichert *)
END;

VAR
  ModuleName : TEXT;    (* Der Name des zu bearbeitenden Moduls *)

  SourceFile : Rd.T;    (* Quelldatei *)
  TargetFile : Wr.T;    (* Zieldatei *)

  SourceText : ModuleText; (* Der Programmtext des Moduls *)
  TargetText : TEXT;       (* Der Text der Interface Datei *)


  ProcHead : ProcedureHead;

(*---------------------------------------------------------------------------
  Name:      IsString

  Aufgabe:   Die Prozedur ueberprueft, ob in einem Text <text> ab der
             aktuellen Position der Text <searchText> steht. <text> ist
             dabei vom Typ ModuleText und verwaltet die aktuelle Position
             als ein Element dieses Records.

  Parameter: text       - der Text in dem gesucht werden soll
             searchText - der gesuchte Text

  Rueckgabe: TRUE / FALSE
---------------------------------------------------------------------------*)
PROCEDURE IsString(text:ModuleText; searchText:TEXT):BOOLEAN =

  VAR offset: CARDINAL;
  BEGIN
    offset := 0; (* Beginne beim ersten Zeichen des gesuchten Textes *)
    LOOP

      (* Wenn das Ende des gesuchten Textes erreicht ist, ist wurde der
         Text gefunden (weil diese Schleife abbricht, sobald
         zwei verschiede Zeichen in den beiden Texten stehen). *)

      IF Text.Length(searchText) = offset THEN
        RETURN TRUE;
      END;

      (* Vergleiche zeichenweise, sind zwei Zeichen verschieden, sind
         auch die Texte verschieden. *)

      IF Text.GetChar(text.txt,text.pos+offset) #
        Text.GetChar(searchText, offset) THEN
        EXIT;
      END;
      INC(offset);
    END;
    RETURN FALSE;
  END IsString;
```

```
(*-------------------------------------------------------------------------------
  Name:       SkipSpaces

  Aufgabe:    Die Prozedur ersetzt alle Leerzeichen, Return- und Newline-Zeichen
              sowie Folgen dieser Zeichen durch ein einzelnes Leerzeichen.
              Steht danach ein Semikolon ';' oder ein oeffnende Klammer '('
              wird kein Leerzeichen eingefuegt. Durch die Bearbeitung des
              Modultextes mit dieser Prozedur erhaelt man eine einheitliche
              Formatierung und muss spaeter keine Sonderfaelle mehr beachten.
              Z.B. kann am Ende einer Procedur <name> stehen:

              END
              <name>     ;

              Nach Aufruf von SkipSpaces wird daraus:

              END <name>;

              Man kann jetzt ganz einfach nach diesem Text suchen.

              Die Prozedur SkipSpaces erhaelt einen Text als Eingabeparameter.
              Sie schaut sich die aktuelle Position des Textes an und entscheidet,
              ob das Zeichen an dieser Position unveraendert uebernommen werden
              soll, z.B. weil es kein Leerzeichen, CR oder NL ist, ob es
              durch ein anderes Zeichen ersetzt werden soll, z.B. ein einzelnes
              NL durch ein Leerzeichen oder ob es uebersprungen werden soll, z.B.
              das erste in einer Reihe von Leerzeichen. Wenn es uebersprungen
              werden soll, geht die Prozedur zum naechsten Zeichen ueber. Das
              Zeichen, das im Ausgabetext stehen soll wird, von der Prozedur
              zurueckgegeben. Die aufrufende Prozedur baut aus den von SkipSpaces
              zurueckgegebenen Zeichen den einheitlich formatierten Text zusammen.

  Parameter: ModuleText - der Eingabetext

  Rueckgabe: das naechste Zeichen des einheitlich formatierten Textes
-------------------------------------------------------------------------------*)
PROCEDURE SkipSpaces (VAR text: ModuleText):TEXT =
VAR result:TEXT;
  BEGIN
    IF IsString(text," ") OR    (* ist aktuelles Zeichen ein Leerzeichen o.ae.?*)
       IsString(text,"\n")  OR
       IsString(text,"\r") THEN

      LOOP
        INC(text.pos);                    (* lies naechstes Zeichen *)
        IF EndOfTextReached(text) THEN    (* Wenn Textende erreicht, gibt "" zurueck *)
          result:="";
          EXIT;

          (* Sonst, wenn kein Leerz. etc. gelesen wird *)

        ELSIF NOT (IsString(text," ") OR IsString(text,"\n") OR IsString(text,"\r")) THEN

          (* Wenn ';' oder ')', dann kein Leerzeichen einfuegen *)

          IF IsString(text,";") OR IsString(text,"(") THEN
            result:= Text.Sub(text.txt,text.pos,1);
            EXIT;
          ELSE

            (* Ansonsten vor dem gelesenen Zeichen ein Leerzeichen einfuegen *)

            result := " " & Text.Sub(text.txt,text.pos,1);
            EXIT;
          END;
        END;
      END;
    ELSE
      (* wenn aktuelles Zeichen kein Leerzeichen o.ae., dann aktuelles
         Zeichen zurueckgeben *)

      result := Text.Sub(text.txt,text.pos,1);
    END;
    INC(text.pos);
    RETURN result;
  END SkipSpaces;
```

```
(*-----------------------------------------------------------------------------
  Name:      SkipComment

  Aufgabe:   Ueberspringt einen Kommentar im Programmtext. Die Prozedur
             schaut nach, ob an der aktuellen Stelle im Text der Anfang
             eines Kommentars steht. Wenn ein solcher gefunden
             wird, ueberspringt die Prozedur den gesamten Text bis
             zum Ende des Kommentars. Zur Bearbeitung geschachtelter
             Kommentare, ruft sich die Prozedur rekursiv wieder auf.

  Parameter: inputText - der Text des Moduls

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE SkipComment(VAR inputText: ModuleText) RAISES {EndOfFileInComment} =
  VAR len: CARDINAL;
  BEGIN
    len := Text.Length(inputText.txt);
    IF IsString(inputText, "(*") THEN
      INC(inputText.pos,2);
      SkipComment(inputText);
      WHILE inputText.pos < len-2 AND NOT IsString(inputText, "*)") DO
        INC(inputText.pos);
        SkipComment(inputText);
      END;
      IF NOT IsString(inputText,"*)") THEN
        RAISE EndOfFileInComment
      ELSE
        INC(inputText.pos,2);
      END;
    END;
  END SkipComment;
(*-----------------------------------------------------------------------------
  Name:      SkipTextLiteral

  Aufgabe:   Ueberspringt eine Textkonstante im Programmtext. Dadurch wird
             verhindert, dass faelschlicherweise Schluesselwoerter oder
             Kommentarzeichen erkannt werden, die in Textkonstanten stehen.

  Parameter: text - der Text des Moduls

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE SkipTextLiteral(VAR text:ModuleText) RAISES {EndOfTextInTextLiteral} =
BEGIN

  (* Wenn an der aktuellen Position eine Zeichenkonstante beginnt *)

  IF Text.GetChar(text.txt, text.pos)='\'' THEN
    INC(text.pos);
    IF Text.GetChar(text.txt, text.pos)='\\' THEN
       INC(text.pos,2);
    ELSE
       INC(text.pos);
    END;
  END;

  (* Wenn an der aktuellen Position eine Textkonstante beginnt *)

  IF Text.GetChar(text.txt, text.pos)='\"' THEN
    INC(text.pos);

    (* Laufe solange durch den Text, bis ein weiteres Anfuehrungszeichen
       kommt, oder der Text zuende ist *)

    WHILE NOT EndOfTextReached(text) AND
       Text.GetChar(text.txt, text.pos) # '\"' DO

        (* beachte keine Zeichen, die auf einen Backslash folgen,
           insbesondere auch keine Anfuehrungszeichen nach
           einem Backslash *)

       IF Text.GetChar(text.txt, text.pos) = '\\' THEN
         INC(text.pos,1);
       END;

       INC(text.pos);
    END;

     (* Wenn das Textende in einer Textkonstante erreicht wurde,
        erzeuge eine Ausnahme. *)

    IF EndOfTextReached(text) THEN RAISE EndOfTextInTextLiteral;
    END;
  END;
END SkipTextLiteral;
```

```
(*---------------------------------------------------------------------------
  Name:      FormatModuleText

  Aufgabe:   Entfernt aus einem Programmtext alle Kommentare und
             ueberfluessigen Leerzeichen etc. Die Prozedur
             verwendet dazu SkipComment und SkipSpaces.

  Parameter: inputText - der Modultext

  Rueckgabe: der formatierte Modultext
---------------------------------------------------------------------------*)
PROCEDURE FormatModuleText(inputText:ModuleText): ModuleText
  RAISES {EndOfFileInComment} =
  VAR
    len:CARDINAL;
    outputText:ModuleText;
    notInTextLit := TRUE;
  BEGIN
    outputText.txt := "";
    inputText.pos  := 0;
    len := Text.Length(inputText.txt);

    (* Im ersten Teil der Prozedur werden die Kommentare entfernt *)

    WHILE NOT EndOfTextReached(inputText) DO

      (* Hier muessen wir nochmal gesondert auf Textkonstanten testen,
         damit wir Kommentarzeichen, die in Textkonstanten auftauchen
         uebergehen koennen. Wir merken uns in der Variablen
         notInTextLit, ob das aktuelle Zeichen ausserhalb einer
         Textkonstanten liegt. Nur wenn das der Fall ist, wird
         ein Kommentarzeichen beachtet. *)

      IF IsString(inputText,"\\") THEN

        (* Anfuehrungszeichen nach einem Backslash ignorieren *)

        outputText.txt := outputText.txt & Text.Sub(inputText.txt,inputText.pos,1);
        INC(inputText.pos);
      ELSIF IsString(inputText,"\"") THEN

        (* Wenn Anf.zeichen gelesen -> wechseln der Zustands *)

        notInTextLit   := NOT notInTextLit;
      ELSIF notInTextLit THEN

        (* Ausserhalb einer Textkonstanten Kommentare beachten *)

        SkipComment(inputText);
      END;
      outputText.txt := outputText.txt & Text.Sub(inputText.txt,inputText.pos,1);
      INC(inputText.pos);
    END;

    (* Im zweiten Teil werden die Leerzeichen und Newline-Zeichen durch
       einzelne Leerzeichen ersetzt *)

    inputText:=outputText;
    outputText.txt := "";
    inputText.pos  := 0;

    WHILE NOT EndOfTextReached(inputText) DO
      outputText.txt := outputText.txt & SkipSpaces(inputText);
    END;

    RETURN outputText;
  END FormatModuleText;
(*---------------------------------------------------------------------------
  Name:      EndOfTextReached

  Aufgabe:   Testet, ob das Textende erreicht ist.

  Parameter: text - der Modultext

  Rueckgabe: TRUE / FALSE
---------------------------------------------------------------------------*)
PROCEDURE EndOfTextReached (text:ModuleText):BOOLEAN =
  BEGIN
    RETURN  text.pos > Text.Length(text.txt)-1;
  END EndOfTextReached;
```

```
(*---------------------------------------------------------------------------
  Name:      FindString

  Aufgabe:   Sucht ab der aktuellen Position nach dem naechsten Vorkommen
             von seachtText in text. Wenn searchText nicht gefunden wird,
             wird die Ausnahmen EndOfText erzeugt.

  Parameter: text       - der Modultext
             searchText - der Text, der gesucht wird

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE FindString(VAR text: ModuleText; searchText: TEXT)
  RAISES {EndOfText, EndOfTextInTextLiteral} =
  BEGIN
    WHILE NOT EndOfTextReached(text) AND
          NOT IsString(text,searchText) DO

      SkipTextLiteral(text);   (* ueberspringe Textkonstanten *)
      INC(text.pos);
    END;
    IF EndOfTextReached(text) THEN RAISE EndOfText;
    END;
  END FindString;
(*---------------------------------------------------------------------------
  Name:      SkipLocalProcedure

  Aufgabe:   Ueberspringt lokale Prozeduren, die genauso heissen, wie die
             globalen Prozeduren, in denen sie definiert sind. Das Ende
             einer globalen Prozedur kann im allgemeinen Fall nicht einfach
             ueber ihren Namen gefunden werden, weil sie lokale
             Unterprozeduren mit dem selben Namen haben kann.

  Parameter: text  - der Modultext
             name  - der Name der globalen Prozedur, deren Ende gefunden werden
                     soll

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE SkipLocalProcedures (VAR text:ModuleText; name:TEXT)
  RAISES {EndOfText, EndOfTextInTextLiteral} =
  BEGIN
    (* Durchlaufe den Text ab der aktuellen Position, bis das
       Ende der globalen Prozedur erreicht ist *)

    WHILE NOT EndOfTextReached(text) AND
          NOT IsString(text, "END " & name & ";") DO

      (* Wenn zwischendurch eine lokale Prozedur mit dem selben
         Namen gefunden wird, such zunaechst deren Ende *)

      IF IsString(text, "PROCEDURE " & name & "(") THEN
        INC(text.pos);
        SkipLocalProcedures(text,name);
      END;
      SkipTextLiteral(text);   (* Ignoriere Textkonstanten *)
      INC(text.pos);
    END;
    IF  EndOfTextReached(text) THEN
      RAISE EndOfText;
    END;
  END SkipLocalProcedures;

(*---------------------------------------------------------------------------
  Name:      FindProcedure

  Aufgabe:   Sucht nach globalen Prozeduren

  Parameter: text - der Modultext

  Rueckgabe: der Kopf und der Name der naechsten globalen Prozedur im Text
---------------------------------------------------------------------------*)
PROCEDURE FindProcedure(VAR text: ModuleText):ProcedureHead
  RAISES {EndOfText, EndOfTextInTextLiteral}=
VAR procedureStart, nameStart, nameEnd: CARDINAL;
    procedureHead: ProcedureHead;
 BEGIN
   FindString(text, "PROCEDURE");  (* Finde Schluesselwort *)

   (* Durch die vorherige Formatierung ist sichergestellt, dass
      der Name der Prozedur vom Schluesselwort PROCEDURE durch
      genau ein Leerzeichen getrennt steht *)

   procedureStart := text.pos;
   nameStart      := procedureStart + Text.Length("PROCEDURE")+1;

   FindString(text, "(");           (* Finde oeffnende Klammer *)

   (* Ebenfalls ist sichergestellt, das die oeffnende Klammer genau
      nach dem Namen steht. *)

   nameEnd            := text.pos;
   procedureHead.name := Text.Sub(text.txt, nameStart,nameEnd - nameStart);
```

```
    (* Finde die schliessende Klammer *)

    FindString(text, ")");

    (* Finde das Gleichzeichen, das den Kopf begrenzt. *)

    FindString(text, "=");

    (* Speichere den Prozedurkopf *)

    procedureHead.txt := Text.Sub(text.txt, procedureStart, text.pos - procedureStart);

    (* Ueberspringe die lokalen Prozeduren  *)

    SkipLocalProcedures(text,procedureHead.name);
    RETURN procedureHead;
  END FindProcedure;

(*---------------------------------------------------------------------------
  Hauptprogramm: Das Hauptprogramm enthaelt zwei ineinander geschachtelte
  Bloecke zur Ausnahmenbehandlung: Der auessere faengt die Ausnahmen der
  importierten Module, wie z.b, SIO ab. Der innere Block behandelt die
  Ausnahmen, die im Modul selbst erzeugt werde, z.B. wenn das Ende des
  Textes erreicht wurde.
---------------------------------------------------------------------------*)
BEGIN
  TRY
    SIO.PutText("\nName der Quelldatei (ohne die Endung '.m3'):");
    ModuleName     := SIO.GetLine();
    SourceText.txt := "";
    TargetText     := "INTERFACE " & ModuleName &";\n\n";

    (* Oeffnen der Dateien *)

    SourceFile := IO.OpenRead(ModuleName & ".m3");
    TargetFile := IO.OpenWrite(ModuleName & ".i3");

    (* Lies die Quelldatei *)

    WHILE NOT IO.EOF(SourceFile) DO
      SourceText.txt := SourceText.txt & "\n" & SIO.GetLine(SourceFile);
    END;
    TRY
      SourceText:=FormatModuleText(SourceText); (* Fomatiere den Text *)

            (* Suche Prozeduren. Die Schleife wird durch das Auftreten
               einer Ausnahme beendet. *)
      LOOP
        ProcHead:= FindProcedure(SourceText);
        SIO.PutText("\n" & ProcHead.txt & ";\n");
        SIO.PutText("   --> Prozedur in das Interface uebernehmen (j/n)?");
        IF Text.Equal(SIO.GetLine(),"j") THEN
          TargetText := TargetText & ProcHead.txt & ";\n"
        END;
      END;
    EXCEPT
    | EndOfText =>
      SIO.PutText("Keine weiteren Prozeduren vorhanden\n");

      TargetText := "\n" & TargetText & "END " & ModuleName & ".";
      SIO.PutLine(TargetText,TargetFile);

    | EndOfTextInTextLiteral =>
      SIO.PutText ("*** Syntaxfehler: Textende beim Lesen einer Stringkonstante ***\n");
    | EndOfFileInComment =>
      SIO.PutText ("*** Syntaxfehler: Textende beim Lesen eines Kommentars ***\n");
    END;
    Rd.Close(SourceFile);
    Wr.Close(TargetFile);
  EXCEPT
  | SIO.Error =>  SIO.PutText("*** Fehler im Modul SIO ***\n");
  | Rd.Failure => SIO.PutText("*** Fehler beim Lesen der Datei ***\n");
  | Wr.Failure => SIO.PutText("*** Fehler beim Schreiben der Datei ***\n");
  | Thread.Alerted =>
  END;
END InterfaceGenerator.
```

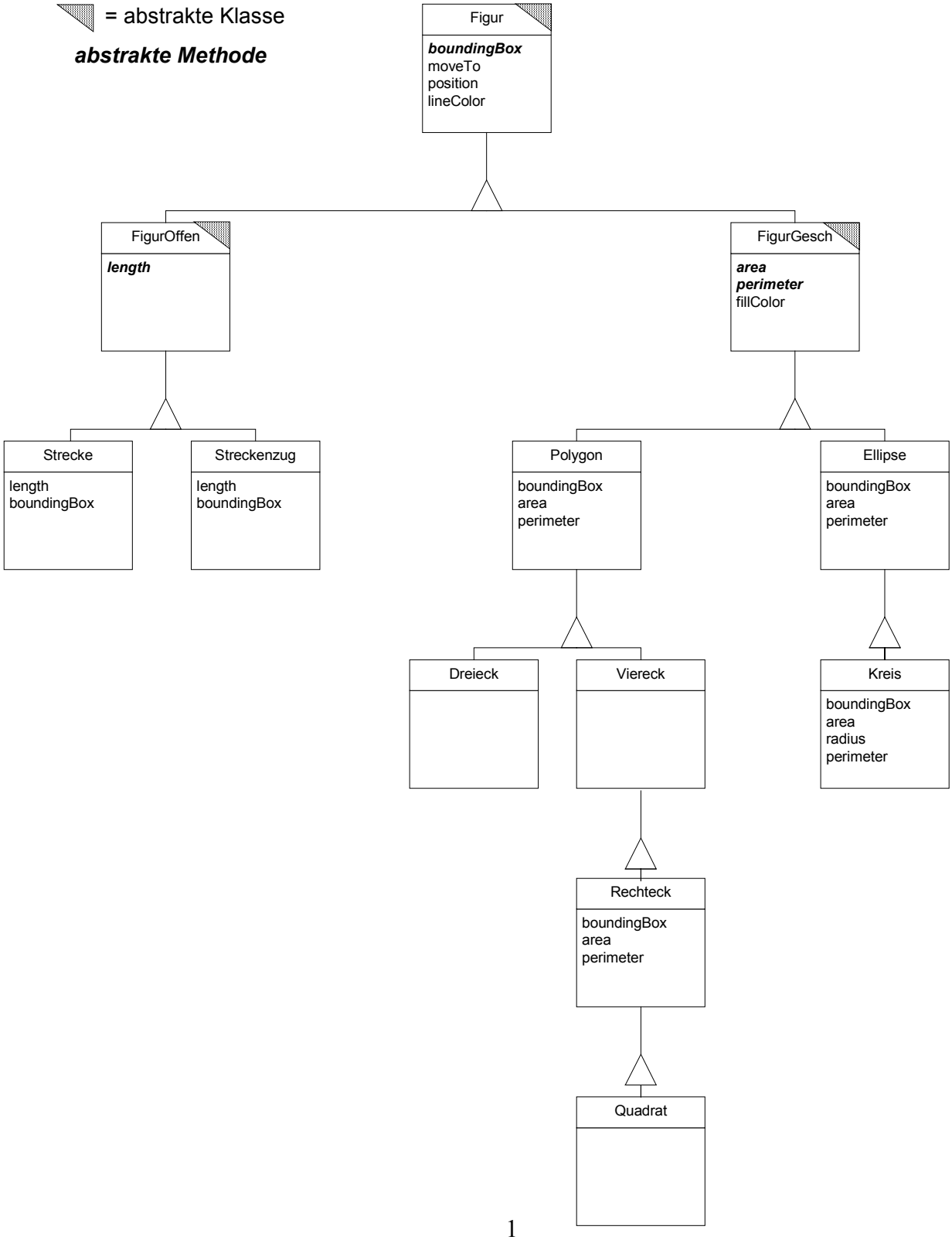
LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **12. Übungsblatt**

Musterlösung

Aufgabe 12.1



Aufgabe 12.2

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.2

  Modul:   TimeCalc
  Datei:   TimeCalc.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE TimeCalc EXPORTS Main;

IMPORT Assertion;

FROM SIO IMPORT PutText, GetInt, PutInt, Nl, Error;

VAR Hrs,Min,dHrs,dMin,newHrs,newMin:INTEGER;
(*---------------------------------------------------------------------------
  Name:      AddTime

  Aufgabe:   Die Prozedur addiert zu einer Uhrzeit eine Zeitspanne und
             liefert als Ergebnis die Uhrzeit nach dem Verstreichen dieser
             Zeitspanne.

  Parameter: hours         - Stunden der aktuellen Uhrzeit
             minutes       - Minuten der aktuellen Uhrzeit
             deltaHours    - Stunden der Zeitspanne
             deltaMinutes  - Minuten der Zeitspanne
             resultHoures  - Rueckgabeparameter: Hier stehen nach Ende der
                             Prozedur die Stunden der Ergebnisuhrzeit.
             resultMinutes - Rueckgabeparameter: Hier stehen nach Ende der
                             Prozedur die Minuten der Ergebnisuhrzeit.

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE AddTime (hours, minutes, deltaHours, deltaMinutes: INTEGER;
           VAR resultHours, resultMinutes: INTEGER)
           RAISES {Assertion.Violated} =

  BEGIN
    Assertion.Require(hours>=0,
                      "Die Stunden der eingegebenen Uhrzeit sind negativ.");
    Assertion.Require(minutes>=0,
                      "Die Minuten der eingegebenen Uhrzeit sind negativ.");
    Assertion.Require(hours<24,
                      "Die Stunden der eingegebenen Uhrzeit sind groesser als 24.");
    Assertion.Require(minutes<60,
                      "Die Minuten der eingegebenen Uhrzeit sind groesser als 60");

    (* Wenn die eingegebenen Minuten groesser oder gleich 60 sind, werden die
       Stunden entsprechend erhoeht *)
    deltaHours   := deltaHours + deltaMinutes DIV 60;

    (* Anschliessend werden alle Minuten groesser oder gleich 60 gekappt *)
    deltaMinutes := deltaMinutes MOD 60;

    (* Die Minuten des Ergebnisses ergeben sich aus den Minuten der Uhrzeit
       plus den zu addierenden Minuten. Werte groesser gleich 60 werden
       hier wieder durch MOD gekappt *)
    resultMinutes:=(minutes + deltaMinutes) MOD 60;

    (* Die Stunden des Ergebnisses ergeben sich aus den Stunden der Uhrzeit
       plus den bei der Berechnung der Minuten gekappten Stunden plus
       den Stunden die zu addieren sind. Auch hier werden Stunden groesser
       oder gleich 24 durch MOD gekappt. (Es war in der Aufgabe ja nicht
       verlangt, dass man ausgibt, bei wieviel Tagen spaeter man landet,
       sondern nur, dass man sagt wie spaet es dann ist. *)
    resultHours  :=(hours + (minutes + deltaMinutes) DIV 60 + deltaHours) MOD 24;
  END AddTime;
```

```
(*-------------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------------*)
BEGIN
  LOOP
    TRY
      PutText("Bitte geben Sie die Stunden der Uhrzeit ein : ");
      Hrs := GetInt();
      PutText("Bitte geben Sie die Minuten der Uhrzeit ein : ");
      Min := GetInt();
      PutText("Wieviele Stunden weiter                    : ");
      dHrs := GetInt();
      PutText("Wieviele Minuten weiter                    : ");
      dMin := GetInt();

      AddTime (Hrs,Min,dHrs,dMin,newHrs,newMin);
      Nl();
      PutText("In ");  PutInt(dHrs);  PutText(" h und ");  PutInt(dMin);
      PutText(" min von ");  PutInt(Hrs);  PutText(".");
      IF newMin<10 THEN PutText("0") END;
      PutInt(Min);
      PutText(" Uhr an gerechnet ist es ");  PutInt(newHrs);  PutText(".");
      IF newMin<10 THEN PutText("0") END;
      PutInt(newMin);  PutText(" Uhr.");  Nl();

      EXIT;  (* Programm wurde ohne Fehler durchlaufen *)
    EXCEPT
    | Error => PutText("Bitte geben Sie nur Integerwerte ein.\n\n");
    | Assertion.Violated =>
      PutText("Bei der Berechung der Uhrzeit ist ein Fehler aufgetreten!\n\n");
    END;
  END; (*LOOP*)
END TimeCalc.
```

Aufgabe 12.3

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.3

  Modul:   AddressMain
  Datei:   AddressMain.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE AddressMain EXPORTS Main;

IMPORT SIO, Text;
FROM AddressOperations IMPORT InputAddress, PrintList, SearchAddress, RemoveName;

VAR input:TEXT;
(*---------------------------------------------------------------------------
  Hauptprogramm
---------------------------------------------------------------------------*)
BEGIN
  TRY
    REPEAT
      SIO.PutText("\n---------------------------------\n");
      SIO.PutText("      M E N U E          \n");
      SIO.PutText("                         \n");
      SIO.PutText(" 1 - Neue Adresse eingeben   \n");
      SIO.PutText(" 2 - Anzeigen der Liste      \n");
      SIO.PutText(" 3 - Suchen nach einem Namen \n");
      SIO.PutText(" 4 - Loeschen eines Elements  \n");
      SIO.PutText(" 5 - Beenden des Programms   \n");
      SIO.PutText("\n\n");
      SIO.PutText("Bitte waehlen Sie eine Funktion: ");
      input:=SIO.GetLine() & " ";
      SIO.Nl();
      SIO.PutText("\n---------------------------------\n");
      SIO.Nl();

      CASE Text.GetChar(input,0) OF
      | '1' => InputAddress();
      | '2' => PrintList();
      | '3' => SearchAddress ();
      | '4' => RemoveName ();
      ELSE END;

    UNTIL Text.GetChar(input,0) ='5';
  EXCEPT SIO.Error => SIO.PutText("*** Fehler im Modul SIO ***\n");
  END;
END AddressMain.



(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 10.1

  Modul:   AddressOperations
  Datei:   AddressOperations.i3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
INTERFACE AddressOperations;

IMPORT SIO;

PROCEDURE PrintList() ;
PROCEDURE InputAddress() RAISES {SIO.Error};
PROCEDURE SearchAddress() RAISES {SIO.Error};
PROCEDURE RemoveName() RAISES {SIO.Error};

END AddressOperations.
```

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.3

  Modul:   AddressOperations
  Datei:   AddressOperations.m3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
MODULE AddressOperations;

IMPORT Address;

FROM AddressStorage IMPORT Init,Insert,Find,Delete,PrintAll;
FROM AddressStorage IMPORT AdrStorage;
FROM AddressStorage IMPORT AddressNotFound;

IMPORT SIO;

VAR Adrst:AdrStorage;  (* zentraler Datenspeicher des ADO *)

(*-----------------------------------------------------------------------------
  Name:      PrintList

  Aufgabe:   gibt die gesamte Liste aus

  Parameter: keine

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE PrintList() =
  BEGIN
    PrintAll(Adrst);
  END PrintList;
(*-----------------------------------------------------------------------------
  Name:      InputAddress

  Aufgabe:   liest eine Adresse vom Benutzer ein und fuegt sie in
             die Liste ein.

  Parameter: keine

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE InputAddress() RAISES {SIO.Error}=
VAR newAdr: Address.T;
  BEGIN
    newAdr := NEW(Address.T);
    SIO.PutText ("Name         : ");
    newAdr.setName(SIO.GetLine());
    SIO.PutText ("Strasse      : ");
    newAdr.setStreet(SIO.GetLine());
    SIO.PutText ("Hausnummer   : ");
    newAdr.setNum(SIO.GetLine());
    SIO.PutText ("Postleitzahl: ");
    newAdr.setPost(SIO.GetLine());
    SIO.PutText ("Ort          : ");
    newAdr.setCity(SIO.GetLine());

    Insert(Adrst,newAdr);
  END InputAddress;
(*-----------------------------------------------------------------------------
  Name:      SearchAddress

  Aufgabe:   liest einen Namen vom Benutzer ein und ruft mit diesem Namen
             der Prozedur Find, auf um danach zu suchen.

  Parameter: keine

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE SearchAddress()  RAISES {SIO.Error} =
 VAR name:TEXT;
     proto,adr:Address.T;

 BEGIN
   SIO.PutText("Nach welchem Namen soll gesucht werden: ");
   name:=SIO.GetLine();
   proto:=NEW(Address.T);
   proto.setName(name);
   TRY
     adr := Find(Adrst,proto);
     SIO.PutText ("Die Adresse lautet: \n\n");
     SIO.PutText (adr.getName()); SIO.Nl();
     SIO.PutText (adr.getStreet() & " " & adr.getNum());  SIO.Nl();
     SIO.PutText (adr.getPost()   & " " & adr.getCity()); SIO.Nl();
     SIO.Nl();
   EXCEPT
   | AddressNotFound => SIO.PutText ("\nDieser Name ist nicht in der Liste!\n")
   END;
 END SearchAddress;
```

```
(*---------------------------------------------------------------------------
  Name:      RemoveName

  Aufgabe:   liest einen Namen vom Benutzer ein und loescht den Eintrag
             mit diesem Namen falls vorhanden.

  Parameter: keine

  Rueckgabe: keine
---------------------------------------------------------------------------*)
PROCEDURE RemoveName()  RAISES {SIO.Error} =
 VAR name:TEXT;
     proto:Address.T;
 BEGIN
   SIO.PutText("Welcher Namen soll geloescht werden: ");
   name:=SIO.GetLine();
   proto := NEW(Address.T);
   proto.setName(name);
   TRY
     Delete(Adrst,proto);
     SIO.PutText ("\nDer Eintrag wurde geloescht!\n")
   EXCEPT
   | AddressNotFound => SIO.PutText ("\nDieser Name ist nicht in der Liste!\n");
   END;
 END RemoveName;
(*---------------------------------------------------------------------------
  Hauptprogramm
---------------------------------------------------------------------------*)
BEGIN
  Init(Adrst);  (* Initialisiere den Datenspeicher *)
END AddressOperations.
```

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.3

  Modul:   Address
  Datei:   Address.i3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
INTERFACE Address;

TYPE
  T<:Public; (* T wird als opaker Untertyp von Public deklariert. *)

 (* Public ist die oeffentliche Schnittstelle des abstrakten Datentyps. *)

  Public = OBJECT
  METHODS
    compare  (adr:T):INTEGER;

    setName  (name: TEXT);
    getName  () : TEXT;
    setStreet(street: TEXT);
    getStreet():TEXT;
    setNum   (num:TEXT);
    getNum   ():TEXT;
    setPost  (post:TEXT);
    getPost  ():TEXT;
    setCity  (city:TEXT);
    getCity  ():TEXT;

    toText():TEXT;
  END;
END Address.
```

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.3

  Modul:   Address
  Datei:   Address.m3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
MODULE Address;

IMPORT Text;

(* Hier wird die interne Struktur des Typs T, der im Interface opak
   deklariert wurde, offengelegt. *)

REVEAL T = Public BRANDED OBJECT

  name:   TEXT;
  street: TEXT;
  num:    TEXT;
  post:   TEXT;
  city:   TEXT;
  OVERRIDES
    compare  := Compare;

    setName  := SetName;
    getName  := GetName;
    setStreet:= SetStreet;
    getStreet:= GetStreet;
    setNum   := SetNum;
    getNum   := GetNum;
    setPost  := SetPost;
    getPost  := GetPost;
    setCity  := SetCity;
    getCity  := GetCity;

    toText   := ToText;
  END;

(*-----------------------------------------------------------------------------
  Name:      Compare

  Aufgabe:   Die Methode vergleicht das Namensfeld der Adresse mit dem Namen
             einer anderen Adresse. Durch Aufruf dieser Funktion koennen
             Adressen in eine Reihenfolge gebracht werden. Wichtig ist dabei,
             dass fuer einen Verwender dieser Funktion nicht ersichtlich ist,
             wie diese Reihenfolge zustande kommt, denn das ist Sache der
             Klasse Address.

  Parameter: adr - ein andere Adresse, die mit dieser Adresse verglichen
                   werden soll

  Rueckgabe: entspricht der Rueckgabe der verwendeten Funktion Text.Compare:
             -1 wenn das Feld name des Objekts (also self.name) alphabetisch
             vor dem Feld name des Objekts adr liegt, 0 wenn beide identisch
             sind und 1 wenn der self.name nach adr.name kommt.
-----------------------------------------------------------------------------*)
PROCEDURE Compare (self: T; adr:T): INTEGER =
  BEGIN
    RETURN Text.Compare(self.name,adr.getName());
  END Compare;
(*-----------------------------------------------------------------------------
  Name:      ToText

  Aufgabe:   Die Methode fasst den Inhalt einer Adresse in einer Zeichenkette
             zusammen und gibt diese Zeichenkette zurueck. Die Funktion
             wird in AddressStorage verwendet, um eine Liste der Adressen
             damit erzeugen zu koennen.

  Parameter: keine

  Rueckgabe: die Adresse als ein Text
-----------------------------------------------------------------------------*)
PROCEDURE ToText (self: T):TEXT =
  BEGIN
    RETURN self.name & ", " & self.street & " " & self.num & ", "
           & self.post & " "& self.city;
  END ToText;
(*-----------------------------------------------------------------------------

  Die folgenden Methoden dienen dem Zugriff auf die internen Daten
  eines Address-Objekts. In diesem Beispiel sind die Methoden sehr einfach,
  sie lesen und schreiben jeweils genau ein Attribut des Objekts.

-----------------------------------------------------------------------------*)

PROCEDURE SetName (self: T; name: TEXT) =
  BEGIN
    self.name := name;
  END SetName;
```

```
PROCEDURE GetName (self: T):TEXT =
  BEGIN
    RETURN self.name
  END GetName;

PROCEDURE SetStreet(self: T; street:TEXT) =
  BEGIN
    self.street := street;
  END SetStreet;

PROCEDURE GetStreet(self: T):TEXT =
  BEGIN
    RETURN self.street;
  END GetStreet;

PROCEDURE SetNum(self: T; num:TEXT) =
  BEGIN
    self.num := num;
  END SetNum;

PROCEDURE GetNum(self: T):TEXT =
  BEGIN
    RETURN self.num;
  END GetNum;

PROCEDURE SetPost(self: T; post:TEXT) =
  BEGIN
    self.post := post;
  END SetPost;

PROCEDURE GetPost(self: T):TEXT =
  BEGIN
    RETURN self.post;
  END GetPost;

PROCEDURE SetCity(self: T; city:TEXT) =
  BEGIN
    self.city := city;
  END SetCity;

PROCEDURE GetCity(self: T):TEXT =
  BEGIN
    RETURN self.city;
  END GetCity;

BEGIN
END Address.
```

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.3

  Modul:   AddressStorage
  Datei:   AddressStorage.i3
  Autor:   Dirk Jaeger

  Im Gegensatz zur vorherigen Loesung wird in AddressStorage nicht mehr
  der Typ Address als ein RECORD deklariert, sondern es wird das Modul
  Address importiert. Die Funktionen der Moduls verwenden nun ausschliesslich
  den Typ Address.T. Auch das Suchen nach einer Addresse funktioniert nicht
  mehr ueber den Namen, sondern ueber die Angabe eines Prototyps, einer
  Adresse, die den selben Namen wie die gesuchte Adresse sonst jedoch keine
  Informationen enthaelt. Vorteil: AddressStorage braucht selber nicht mehr
  zu wissen, was das Suchkriterium. Dieses Wissen muss nur in Address
  vorhanden sein, denn dort wird der Vergleich implementiert, und in
  dem verwendenden Modul, denn dort muss der entsprechende Prototyp
  vorhanden sein. Durch diese Trennung ist es dann im naechsten Schritt
  (Aufgabenblatt 13) moeglich, AddressStorage zu einem allgemeinen
  Objektspeicher zu verallgemeinern, der an die zu speichernden Objekte nur
  noch die Anforderung stellt, dass sie die Methoden compare und toText
  implementieren.
---------------------------------------------------------------------------*)
INTERFACE AddressStorage;

IMPORT Address;

TYPE
  AdrStorage <: REFANY;

EXCEPTION AddressNotFound;

PROCEDURE Init    (VAR adrst:AdrStorage) ;
PROCEDURE Insert  (VAR adrst:AdrStorage; newAdr:Address.T) ;
PROCEDURE Find    (    adrst:AdrStorage; proto:Address.T): Address.T RAISES {AddressNotFound};
PROCEDURE Delete  (VAR adrst:AdrStorage; proto:Address.T) RAISES {AddressNotFound};
PROCEDURE PrintAll(    adrst:AdrStorage);

END AddressStorage.

(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 12.3

  Modul:   AddressStorage
  Datei:   AddressStorage.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE AddressStorage;

IMPORT SIO, Address;

(* Hier wird definiert, auf was der opake Referenztyp AdrStorage
   tatsaechlich verweist *)

REVEAL AdrStorage = BRANDED REF ListType;

TYPE
  Element = RECORD     (* der Typ eines Listenelements *)
    adr:  Address.T;
    next: REF Element;
    prev: REF Element;
  END;

  ListType = RECORD    (* der Typ der Liste *)
    head: REF Element; (* Zeiger auf das erste Element *)
    tail: REF Element; (* Zeiger auf das letzte Element *)
  END;
(*---------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Die Prozedur initialisiert den ADT, der durch eine doppelt
             verkettete Liste realisiert wird. Damit man beim Einfuegen
             und Loeschen nicht die Sonderfaelle (leere Liste, einfuegen am
             Ende, einfuegen am Anfang) behandeln muss, wird die Liste
             initial mit einem vordersten Element (head) und einem
             hintersten Element (tail) belegt. Diese beiden Element markieren
             den Anfang und das Ende der Liste. Sie enthalten keine Daten
             und werden nie geloescht. Neue Elemente werden deshalb immer
             zwischen zwei bereits vorhandenen Listenelementen eingefuegt,
             Sonderfaelle gibt es nicht.

  Parameter: adrst - eine Instanz des ADT

  Rueckgabe: keine
---------------------------------------------------------------------------*)
```

```
PROCEDURE Init(VAR adrst:AdrStorage) =
  BEGIN
    adrst := NEW(AdrStorage);
    adrst^.head := NEW (REF Element);
    adrst^.tail := NEW (REF Element);
    adrst^.head^.next := adrst^.tail;
    adrst^.head^.prev := NIL;
    adrst^.head^.adr := NEW(Address.T);
    adrst^.head^.adr.setName("HEAD");
    adrst^.head^.adr.setStreet("---");
    adrst^.head^.adr.setNum("---");
    adrst^.head^.adr.setPost("---");
    adrst^.head^.adr.setCity("---");

    adrst^.tail^.prev := adrst^.head;
    adrst^.tail^.next := NIL;
    adrst^.tail^.adr := NEW(Address.T);
    adrst^.tail^.adr.setName("TAIL");
    adrst^.tail^.adr.setStreet("---");
    adrst^.tail^.adr.setNum("---");
    adrst^.tail^.adr.setPost("---");
    adrst^.tail^.adr.setCity("---");
  END Init;
(*-----------------------------------------------------------------------------
  Name:      Insert

  Aufgabe:   fuegt eine neue Adresse in die Liste ein. Die Liste wird solange
             durchlaufen, wie das aktuellen Listenelements kleiner als
             das einzufuegenden Elements ist. Zum Vergleich der Elemente wird
             die Methode compare verwendet.
             Wenn diese Bedingung nicht mehr gilt, gibt es zwei Faelle:
             Entweder die beiden Elemente sind gleich, dann soll laut
             Aufgabenstelleung der alten Eintrag einfach durch den neuen
             ueberschrieben werden, oder die beiden Elemente sind nicht
             gleich, dann wird ein neues Element in die Liste eingefuegt.

  Parameter: adrst  - eine Instanz des ADT
             newAdr - die neue Adresse.

  Rueckgabe: keine
-----------------------------------------------------------------------------*)
PROCEDURE Insert(VAR adrst:AdrStorage; newAdr:Address.T) =
  VAR
    newPointer, helpPointer : REF Element;

  BEGIN
    helpPointer:=adrst^.head;

   (* Suche die Stelle zum Einfuegen in die Liste. *)

    WHILE helpPointer^.next^.adr.compare(newAdr) <= 0 AND
          helpPointer^.next # adrst^.tail DO

      helpPointer:=helpPointer^.next;
    END;

    (* Pruefe ob die beiden Elemente gleich sind *)
    IF helpPointer^.adr.compare(newAdr) = 0 THEN

      helpPointer^.adr := newAdr;   (* alte Adresse wird ersetzt *)
    ELSE
      newPointer := NEW (REF Element);  (* neues Element anlegen und einfuegen *)
      newPointer^.adr := newAdr;
      newPointer^.next := helpPointer^.next;
      newPointer^.prev := helpPointer;
      helpPointer^.next^.prev := newPointer;
      helpPointer^.next := newPointer;
    END;
  END Insert;

(*-----------------------------------------------------------------------------
  Name:      FindElement

  Aufgabe:   Die Prozedur sucht anhand eines Prototyps nach einem Listeneintrag.
             Die Liste wird solange durchlaufen, bis entweder der gesuchte
             Eintrag gefunden wurde, oder der aktuelle Eintrag
             groesser ist, als der gesuchte, oder das Ende der Liste erreicht
             wurde.

  Parameter: adrst - eine Instanz des ADT
             proto - eine Adresse, die denselben Schluessel hat, wie die
                     gesuchte Adresse

  Rueckgabe: einen Zeiger auf das gesuchte Element. Falls das gesuchte
             Element nicht in der Liste ist, wird NIL zurueckgegeben
-----------------------------------------------------------------------------*)
```

```
PROCEDURE FindElement (adrst:AdrStorage; proto:Address.T): REF Element=
VAR  helpPointer : REF Element;

BEGIN
  helpPointer := adrst^.head^.next;

  (* Suche nach einem Element mit demselben Schluessel *)
  WHILE (helpPointer^.adr.compare(proto) < 0) AND
    (helpPointer # adrst^.tail) DO
    helpPointer:= helpPointer^.next;
  END;

  (* Wenn Listenende erreicht -> NIL *)
  IF helpPointer = adrst^.tail THEN RETURN NIL;
  ELSE

    (* Wurde das Element gefunden? *)
    IF helpPointer^.adr.compare(proto) = 0 THEN

      RETURN helpPointer; (* Zeiger auf das gesuchte Element *)
    ELSE
      RETURN NIL;         (* sonst NIL zurueckgeben *)
    END;
  END;
END FindElement;
(*-------------------------------------------------------------------------
  Name:      Find

  Aufgabe:   Die Prozedur sucht nach einem Namen in der Liste und gibt
             die Adresse zu diesem Namen zurueck. Wenn der Name nicht
             in der Liste ist, wird eine Ausnahme erzeugt

  Parameter: adrst - eine Instanz des ADT
             proto - eine Adresse, die denselben Schluessel hat, wie die
                     gesuchte Adresse

  Rueckgabe: die Adresse zu dem gesuchten Namen
-------------------------------------------------------------------------*)
PROCEDURE Find(adrst:AdrStorage; proto:Address.T): Address.T RAISES {AddressNotFound} =
VAR pointer: REF Element;
BEGIN
  pointer := FindElement(adrst,proto);
  IF pointer # NIL THEN
    RETURN pointer^.adr;
  ELSE
    RAISE AddressNotFound;
  END;
END Find;
(*-------------------------------------------------------------------------
  Name:      Delete

  Aufgabe:   Die Prozedur loescht ein Element aus der Liste indem sie
             es aus der Listenstruktur auskettet.

  Parameter: adrst - eine Instanz des ADT
             proto - eine Adresse, die denselben Schluessel hat, wie die
                     gesuchte Adresse
  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Delete (VAR adrst:AdrStorage; proto:Address.T) RAISES {AddressNotFound}=
VAR pointer: REF Element;
  BEGIN
    pointer := FindElement(adrst,proto);
    IF pointer # NIL THEN
      pointer^.prev^.next := pointer^.next;
      pointer^.next^.prev := pointer^.prev;
      pointer := NIL;
    ELSE
      RAISE AddressNotFound;
    END;
  END Delete;
(*-------------------------------------------------------------------------
  Name:      PrintAll
  Aufgabe:   gibt die gesamte Liste aus
  Parameter: adrst - eine Instanz des ADT
  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintAll(adrst:AdrStorage) =
VAR helpPointer: REF Element;
  BEGIN
    helpPointer := adrst^.head;
    WHILE helpPointer # NIL DO
      SIO.PutText( helpPointer^.adr.toText());
      SIO.Nl();
      helpPointer:= helpPointer^.next;
    END;
  END PrintAll;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
END AddressStorage.
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja*|*jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **13. Übungsblatt**

Musterlösung

Aufgabe 13.1

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.1

  Modul:   Grafik
  Datei:   Grafik.m3
  Autor:   Dirk Jaeger

  Das Modul implementiert die Klassenhierarchie aus Aufgabe 12.1. Durch die
  grosse Anzahl von Klassen, mit jeweils nur wenigen Methoden, ist eine
  Implementierung in der ueblichen Weise, eine Klasse pro Datei, sehr
  unuebersichtlich. Aus diesem Grund wurden alle Klassen in einer
  Datei implementiert. Ebenfalls aus Gruenden der Vereinfachung wurde auf
  das Information Hiding durch Definition von opaken Klassen verzichtet.

  Da die Methoden aller Klassen in einem Modul stehen, erhalten die Namen
  der einzelnen Methoden zur Unterscheidung jeweils als Praefix den
  Klassennamen.
-------------------------------------------------------------------------*)
MODULE Grafik EXPORTS Main;

FROM Math IMPORT sqrt, pow, Pi;

CONST
  MAXPOINTS = 100;

TYPE
  Color     = {RED, GREEN, BLUE, WHITE, BLACK}; (* Farben: Nur als Beispiel *)

  (* An vielen Stellen der Klassenhierarchie muss man mit Listen von Punkten
     arbeiten. Hier stellt sich das uebliche Problem, dass man eigentlich
     eine dynamische Datenstruktur haben muesste um auf der einen Seite eine
     beliebige Anzahl von Punkten verwalten zu koennen, und andererseits
     bei wenigen Punkten (Dreicke, Vierecke) keine Speicherplatz zu
     verschwenden. Um das Programm zu vereinfachen, habe ich hier als
     Punktliste einen Array von MAXPOINTS(=100) + 1 Punkten definiert.
     Das zusaetzliche Feld in der List vereinfacht die Implementierung
     von Berechnungen fuer Polygone, weil man in diesem Feld den ersten
     Punkt des Polygons nochmals hinten an die Liste anhaengen kann. *)

  PointList = ARRAY[1..MAXPOINTS+1] OF Point;

(*-------------------------------------------------------------------------
  Klasse Point

  Die Klasse ist nicht Teil der Hierarchie, sie wird aber zu deren
  Realisierung benoetigt. Die Methode dist berechet die Entfernung des
  Punktes zu einem anderen Punkt.
-------------------------------------------------------------------------*)
TYPE
  Point = OBJECT
    x,y: LONGREAL;

  METHODS
    dist(p:Point):LONGREAL := Dist;
  END;

PROCEDURE Dist(self: Point; p: Point): LONGREAL =
  BEGIN
    RETURN sqrt(pow(self.x - p.x,2.0D0) + pow(self.y - p.y,2.0D0));
  END Dist;
```

```
(*-------------------------------------------------------------------------
  Klasse Figure

  Die Klasse Figure ist die abstrakte Oberklasse aller Grafikobjekt.
  In der Klasse wird die Position und die Farbe definiert. Die Methoden
  moveTo, position und lineColor schreiben und lesen diese Attribute.
  Weiterhin wird die Methode boundingBox abstrakt definiert. Die Methode
  liefert ein Objekt der Klasse Rectangle zurueck.

  Die Position des Objekts wird einfach in dem entsprechenden Attribut
  abgelegt. Die Annahme ist, das ein Benutzer der Klassenhierarchie die
  Punkte, durch die ein Objekt definiert ist, relativ zu der Position
  des Objekts interpretiert. Fuer die Berechnungen spielt die Position
  keine Rolle, da Flaeche und Umfang eines Objekts von der Position
  unabhaengig sind. Bei Berechnung der Bounding Box wird als Position
  der Bounding Box die Position des Objekts uebernommen.

  Fuer die Berechnung der Bounding Box und der Laenge eines Streckenzugs
  werden in der Klasse Figure zwei Hilfsmethoden definiert, die in den
  Unterklassen Polygon und Polyline verwendet werden.
-------------------------------------------------------------------------*)
TYPE
  Figure = OBJECT
    Position: Point;
    LineColor: Color;
  METHODS
    boundingBox():Rectangle;
    moveTo(p:Point)    := FigureMoveTo;
    position():Point   := FigurePosition;
    lineColor(c:Color) := FigureLineColor;

    calculateBBox  (VAR points: PointList; numPoints: CARDINAL;
                    position:Point):Rectangle     := FigureCalculateBBox;
    calculateLength(VAR points: PointList;
                    numPoints: CARDINAL):LONGREAL := FigureCalculateLength;
  END;

PROCEDURE FigureMoveTo(self:Figure; p:Point) =
  BEGIN self.Position:=p; END FigureMoveTo;

PROCEDURE FigurePosition(self:Figure):Point =
  BEGIN RETURN self.Position; END FigurePosition;

PROCEDURE FigureLineColor(self:Figure; c:Color) =
  BEGIN self.LineColor:=c; END FigureLineColor;

PROCEDURE FigureCalculateBBox(<*UNUSED*>self: Figure; VAR points: PointList;
                              numPoints:CARDINAL; position:Point):Rectangle=
  VAR bbox:Rectangle;
  BEGIN
    bbox := NEW (Rectangle, Points := points);
    FOR i:= 2 TO numPoints DO
      IF points[i].x > bbox.Points[2].x THEN bbox.Points[2].x := points[i].x END;
      IF points[i].x < bbox.Points[1].x THEN bbox.Points[1].x := points[i].x END;
      IF points[i].y < bbox.Points[2].y THEN bbox.Points[2].y := points[i].y END;
      IF points[i].y > bbox.Points[1].y THEN bbox.Points[1].y := points[i].y END;
    END;
    bbox.moveTo(position);
    RETURN bbox;
  END FigureCalculateBBox;

PROCEDURE FigureCalculateLength(<*UNUSED*> self: Figure; VAR points: PointList;
                                numPoints: CARDINAL):LONGREAL =
  VAR length:LONGREAL;
  BEGIN
    FOR i:=1 TO numPoints-1 DO
      length := length + points[i].dist(points[i+1]);
    END;
    RETURN length
  END FigureCalculateLength;
(*-------------------------------------------------------------------------
  Klasse OpenFigure

  Offene Grafikobjekte besitzen eine Methode zur Berechnung ihrer Laenge
-------------------------------------------------------------------------*)
TYPE
  OpenFigure = Figure OBJECT
  METHODS
    length():LONGREAL;
  END;
```

```
(*-------------------------------------------------------------------------
  Klasse Linie

  Eine Linie wird durch zwei Punkte definiert. Die Laenge einer Linie
  ergibt sich durch den Abstand der Endpunkte.

  Die Bounding Box einer Linie berechnen wir mit der Hilfsmethode
  calculateBoundingBox. Die Methode erhaelt als Parameter eine Liste aus
  zwei Punkten.
-------------------------------------------------------------------------*)
TYPE
  Line = OpenFigure OBJECT
    From,To: Point;
  OVERRIDES
    length      := LineLength;
    boundingBox := LineBoundingBox;
  END;

PROCEDURE LineLength(self:Line):LONGREAL =
  BEGIN
    RETURN self.From.dist(self.To);
  END LineLength;

PROCEDURE LineBoundingBox(self:Line):Rectangle =
  VAR points:PointList;
  BEGIN
    points[1]:=self.From;
    points[2]:=self.To;
    RETURN self.calculateBBox(points,2, self.Position);
  END LineBoundingBox;

(*-------------------------------------------------------------------------
  Klasse PolyLine

  Die Klasse Polyline wird durch eine Liste von Punkte implementiert.
  Mit dieser Liste koennen dann direkt die Methoden calculateLength und
  calculateBoundingBox aufgerufen werden.
-------------------------------------------------------------------------*)
TYPE
  PolyLine = OpenFigure OBJECT
    NumOfPoints: CARDINAL;
    Points:  PointList;
  OVERRIDES
    length      := PolyLineLength;
    boundingBox := PolyLineBoundingBox;
  END;

PROCEDURE PolyLineLength(self: PolyLine):LONGREAL =
  BEGIN
    RETURN self.calculateLength(self.Points, self.NumOfPoints);
  END PolyLineLength;

PROCEDURE PolyLineBoundingBox(self: PolyLine):Rectangle =
  BEGIN
    RETURN self.calculateBBox(self.Points, self.NumOfPoints, self.Position);
  END PolyLineBoundingBox;

(*-------------------------------------------------------------------------
  Klasse ClosedFigure

  Bei einer geschlossenen Figur kommen ein Attribut fuer die Fuellfarbe
  und abstrakte Methoden zur Berechung des Umfangs und der Flaeche hinzu.
-------------------------------------------------------------------------*)
TYPE
  ClosedFigure = Figure OBJECT
    FillColor: Color;
  METHODS
    fillColor(c:Color):= ClosedFigureFillColor;
    area():LONGREAL;
    perimeter():LONGREAL;
  END;

PROCEDURE ClosedFigureFillColor(self: ClosedFigure; c:Color) =
BEGIN
  self.FillColor:= c;
END  ClosedFigureFillColor;
```

```
(*-------------------------------------------------------------------------
  Klasse Polygon

  Ein Polygon wird durch eine Liste von Eckpunkten definiert. In NumOfPoints
  wird angegeben, wieviele Punkte (Ecken) das Polygon besitzt.
  Diese Datenstruktur kann auch fuer die von Polygon abgeleiteten Klassen
  uebernommen werden.

  Fuer die Flaeche eines Polygons gibt es eine einfache Formel, die allerdings
  voraussetzt, dass sich ein Polygon nicht selbst ueberschneidet.
  Eine Moeglichkeit, diesen Fall zu behandeln ist, durch Einfuegen von zwei
  zusaetzlichen (uebereinanderliegenden) Eckpunkten an den
  Ueberschneidungsstellen um Umordnen der Punkte, das Polygon in ein
  ueberschneidungsfreies umzuwandeln (hier nicht implementiert).
  Weiterhin setzt das Verfahren voraus, dass die Eckpunkte in der Liste
  im Uhrzeigersinn angeordnet sind.

  Die Berechnung der Bounding Box und des Umfangs stuetzen sich wieder
  auf die Methoden von Figure ab.
-------------------------------------------------------------------------*)
TYPE
  Polygon = ClosedFigure OBJECT
    NumOfPoints : CARDINAL;
    Points: PointList;
  OVERRIDES
    area        := PolygonArea;
    boundingBox := PolygonBoundingBox;
    perimeter   := PolygonPerimeter;
  END;

PROCEDURE PolygonArea(self: Polygon):LONGREAL =
  VAR area: LONGREAL:=0.0D0;
  BEGIN
    self.Points[self.NumOfPoints+1] := self.Points[1];
    FOR i:=1 TO self.NumOfPoints DO
      area := area + self.Points[i].y*(self.Points[i+1].x - self.Points[i].x);
    END;
    RETURN area;
  END PolygonArea;

PROCEDURE PolygonBoundingBox(self: Polygon):Rectangle =
  BEGIN
    RETURN self.calculateBBox(self.Points, self.NumOfPoints, self.Position);
  END PolygonBoundingBox;

PROCEDURE PolygonPerimeter (self: Polygon):LONGREAL =
  BEGIN
    self.Points[self.NumOfPoints+1]:= self.Points[1];
    RETURN self.calculateLength(self.Points, self.NumOfPoints+1);
  END PolygonPerimeter;

(*-------------------------------------------------------------------------
  Klasse Triangle

  Ein Dreieck ist ein Polygon mit drei Punkten. Man kann also die selben
  Methoden verwenden wie fuer Polygon. In einer vollstaendig ausimplementierten
  Klassehierarchie muesste man natuerlich sicherstellen, dass in der
  Punktliste von Triangle immer genau drei Punkte stehen.
-------------------------------------------------------------------------*)
TYPE
  Triangle = Polygon OBJECT
  END;

(*-------------------------------------------------------------------------
  Klasse Quadrangle

  Das selbe wie fuer Triangle gilt auch fuer die Klasse Quadrangle
-------------------------------------------------------------------------*)
TYPE
  Quadrangle = Polygon OBJECT
  END;
```

```
(*-------------------------------------------------------------------------
  Klasse Rectangle

  In der Klasse Rectangle verwenden wir zwar weiter die Datenstruktur, die
  in Polygon definiert wurde, wir interpretieren sie aber anders. Die
  Punktliste besteht bei Reactangle aus genau zwei Punkten, von denen
  der erste die linke obere Ecke und der zweite die rechte untere Ecke des
  Rechtecks bezeichnet.

  Als Bounding Box gibt das Rechteck sich selbst zurueck.
  Die Flaeche und der Umfang eines Rechtecks werden als Produkt bzw. Summe
  der Seitlaengen mal zwei berechnet.
-------------------------------------------------------------------------*)
TYPE
  Rectangle = Quadrangle OBJECT
  OVERRIDES
    boundingBox := RectangleBoundingBox;
    area        := RectangleArea;
    perimeter   := RectanglePerimeter;
  END;

PROCEDURE RectangleBoundingBox (self: Rectangle):Rectangle =
 BEGIN
   RETURN self;
 END RectangleBoundingBox;

PROCEDURE RectangleArea (self: Rectangle):LONGREAL =
  BEGIN
    RETURN (self.Points[2].x - self.Points[1].x) *
           (self.Points[1].y - self.Points[2].y);
  END RectangleArea;

PROCEDURE RectanglePerimeter (self: Rectangle):LONGREAL =
  BEGIN
    RETURN (self.Points[2].x - self.Points[1].x) *
           (self.Points[1].y - self.Points[2].y) * 2.0D0;
  END RectanglePerimeter;

(*-------------------------------------------------------------------------
  Klasse Square

  Die Klasse Square definiert keine eigenen Methoden.
-------------------------------------------------------------------------*)
TYPE
  Square = Rectangle OBJECT
  END;

(*-------------------------------------------------------------------------
  Klasse Ellipse

  Als Datenstruktur einer Ellipse werden die beiden Brennpunkte F1 und F2, der
  Durchmesser a entlang der grossen Achse  (= 2 * der maximale Radius) und
  der Durchmessern b entlang der kleinen Achse verwaltet. Der Durchmesser
  entlang der kleinen Achse war in der Aufgabenstellung nicht vorgegeben,
  die Speicherung dient nur dazu, ihn nicht jedesmal neu berechnen zu
  muessen. Er kann aus dem Durchmesser der grossen Achse ueber die Beziehung:

            b = sqrt( a^2 - Abstand(F1,F2)^2)

  berechnet werden.

  Mit diesen Werten koennen die Flaeche (exakt) und der Umfang ueber
  eine Naeherungsformel (siehe mathematische Formelsammlung) berechnet werden.

  Die Berechnung der Bounding Box ist dagegen wesentlich komplizierter und
  wird hier nur skizziert:

  Man geht von der Parameterdarstellung der Ellipse aus:

  x = a cos(t)
  y = b sin(t)

  Diese Darstellung beschreibt eine Ellipse, deren Achsen auf dem
  Koordinatenachsen liegen und deren Mitte im Ursprung des Koordinatensystems
  liegt.

  Die Parameterdarstellung der tatsaechlich vorliegenden Ellipse erhaelt man,
  durch Drehen und Verschieben. Man dreht die Ellipse durch Multiplikation
  mit einer Rotationsmatrix A mit

       cos(alpha)  sin(alpha)
  A =
      -sin(alpha)  cos(alpha)

  Dabei ist alpha der Winkel zwischen der grossen Achse der Ellipse und
  der x-Koordinate.

  Als neue Parameterdarstellung erhalt man durch Ausmultiplizieren:

  x =   a cos(t) cos(alpha) + b sin(t) sin (alpha)
  y = - a cos(t) sin(alpha) + b sin(t) cos (alpha)

  Von beiden Komponenten bildet man nun die Ableitung nach t und erhaelt so
```

```
  die Nullstellen. Bei diesen Parameterwerten fuer t ist die Zunahme der
  x bzw. y Position gleich null, d.h. es sind die auesserst linken und rechten,
  bzw. oberen und unteren Punkte der Ellipse. Man setzt die Werte fuer
  t in die Paremeterdarstellung ein und berechnet die zugehoerigen
  Koordinaten. Damit hat man die Koordinaten der Bounding Box, deren
  Mitte allerdings noch im Ursprung des Koordinatensystems liegt.
  Man muss jetzt also zu den Koordinaten noch den tatsaechlichen
  Mittelpunkt der Ellipse hinzuaddieren.
-------------------------------------------------------------------------*)
TYPE
  Ellipse = ClosedFigure OBJECT
    F1,F2: Point;
    a,b: LONGREAL;
  OVERRIDES
    boundingBox := EllipseBoundingBox;
    area        := EllipseArea;
    perimeter   := EllipsePerimeter;
  END;

PROCEDURE EllipseBoundingBox(<*UNUSED*>self:Ellipse): Rectangle =
  VAR rect:Rectangle;
  BEGIN
    rect:=NEW (Rectangle);

    (* nicht implementiert: Skizze siehe oben *)

    RETURN rect;
  END EllipseBoundingBox;

PROCEDURE EllipseArea(self:Ellipse): LONGREAL =
  BEGIN
    RETURN FLOAT(Pi,LONGREAL) * self.a * self.b;
  END EllipseArea;

PROCEDURE EllipsePerimeter(self: Ellipse): LONGREAL =
  BEGIN
    RETURN FLOAT(Pi,LONGREAL) * ( 1.5D0 * (self.a + self.b) - sqrt(self.a * self.b));
  END EllipsePerimeter;

(*-------------------------------------------------------------------------
  Klasse Circle

  Die Klasse verwendet die ueblichen Formel fuer die Berechnung von
  Umfang und Radius. Es werden die in Ellipse definierten
  Datenstrukturen uebernommen. F1 ist der Mittelpunkt und a ist der Durchmesser.
-------------------------------------------------------------------------*)
TYPE
  Circle = Ellipse OBJECT
  METHODS
    radius():LONGREAL := CircleRadius;

  OVERRIDES
    boundingBox := CircleBoundingBox;
    area        := CircleArea;
    perimeter   := CirclePerimeter;
  END;

PROCEDURE CircleBoundingBox(self: Circle): Rectangle =
VAR bbox:Rectangle;
  BEGIN
    bbox:=NEW (Rectangle);
    bbox.Points[1].x  := self.F1.x - self.a / 2.0D0;
    bbox.Points[1].y  := self.F1.y - self.a / 2.0D0;
    bbox.Points[2].x  := self.F1.x + self.a / 2.0D0;
    bbox.Points[2].y  := self.F1.y - self.a / 2.0D0;
    RETURN bbox;
  END CircleBoundingBox;

PROCEDURE CircleArea(self: Circle): LONGREAL =
  BEGIN
    RETURN FLOAT(Pi,LONGREAL) * pow(self.a / 2.0D0,2.0D0);
  END CircleArea;

PROCEDURE CirclePerimeter(self: Circle): LONGREAL =
  BEGIN
    RETURN FLOAT (Pi,LONGREAL) * self.a;
  END CirclePerimeter;

PROCEDURE CircleRadius(self:Circle): LONGREAL =
  BEGIN
    RETURN self.a / 2.0D0;
  END CircleRadius;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
END Grafik.
```

Aufgabe 13.2

a)

```
(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   StorableObject
  Datei:   StorableObject.i3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
INTERFACE StorableObject;

TYPE
  T<:Public;

  (* Definiere eine allgemeine Klasse von Objekten, die in
     einem Objektspeicher abgelegt werden koennen. Diese
     Objekt muessen die beiden Methoden compare und toText
     implementieren *)

  Public = OBJECT
  METHODS
    compare(otherObject:T):INTEGER;   (* vergleiche zwei Objekte *)
    toText ():TEXT;                   (* gib Objekt als Text aus *)
  END;

END StorableObject.



(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   StorableObjekt
  Datei:   StorableObjekt.m3
  Autor:   Dirk Jaeger

  Dieses Modul stellt eine Default Implementierung fuer speicherbare
  Objekte zur Verfuegung. Damit koennen Dummy-Objekte dieser Klasse
  erzeugt werden, so wie sie z.B. im Modul ObjectStorage zum
  Markieren von Listenanfang und Listenende verwendet werden.
---------------------------------------------------------------------------*)
MODULE StorableObject;

REVEAL
  T= Public BRANDED OBJECT
  OVERRIDES
    compare:=Compare;
    toText:=ToText;
  END;

PROCEDURE Compare(<*UNUSED*>self:T; <*UNUSED*>otherObject:T):INTEGER=
  BEGIN
    RETURN -1
  END Compare;

PROCEDURE ToText(<*UNUSED*>self:T):TEXT =
  BEGIN
    RETURN "--------------------";
  END ToText;

BEGIN
END StorableObject.
```

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   Address
  Datei:   Address.i3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
INTERFACE Address;

IMPORT StorableObject;

TYPE
  T<:Public; (* T wird als opaker Untertyp von Public deklariert. *)


  (* Public ist die oeffentliche Schnittstelle des abstrakten Datentyps.
     Public wird als Unterklasse von StorableObject deklariert.
     In der Implementierun muessen dann die Methoden von StorableObjekt.T
     durch eigene Methoden der Klasse ueberschrieben werden.
     Objekte der Klasse Address.T koennen damit ueberall dort verwendet
     werden, wo Objekte der Klasse StorableObjekt verlangt sind,
     z.B. als Methoden der Parameter der Klasse ObjectStorage*)

  Public = StorableObject.T OBJECT
  METHODS
    setName  (name: TEXT);
    getName  () : TEXT;
    setStreet(street: TEXT);
    getStreet():TEXT;
    setNum   (num:TEXT);
    getNum   ():TEXT;
    setPost  (post:TEXT);
    getPost  ():TEXT;
    setCity  (city:TEXT);
    getCity  ():TEXT;
  END;
END Address.



(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   Address
  Datei:   Address.m3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
MODULE Address;

IMPORT Text;
IMPORT StorableObject;

(* Hier wird die interne Struktur des Typs T, der im Interface opak
   deklariert wurde, offengelegt. *)

REVEAL T = Public BRANDED OBJECT

  name:   TEXT;
  street: TEXT;
  num:    TEXT;
  post:   TEXT;
  city:   TEXT;
  OVERRIDES
    compare  := Compare;

    setName  := SetName;
    getName  := GetName;
    setStreet:= SetStreet;
    getStreet:= GetStreet;
    setNum   := SetNum;
    getNum   := GetNum;
    setPost  := SetPost;
    getPost  := GetPost;
    setCity  := SetCity;
    getCity  := GetCity;

    toText   := ToText;
  END;
```

```
(*-----------------------------------------------------------------------------
  Name:       Compare

  Aufgabe:    Die Methode vergleicht das Namensfeld der Adresse mit dem Namen
              einer anderen Adresse. Durch Aufruf dieser Funktion koennen
              Adresse in eine Reihenfolge gebracht werden. Wichtig ist dabei,
              dass fuer einen Verwender dieser Funktion nicht ersichtlich ist,
              wie diese Reihenfolge zustande kommt, denn das ist Sache der
              Klasse Address.

  Parameter: adr - ein andere Adresse, die mit dieser Adresse verglichen
                   werden soll

  Rueckgabe: entspricht der Rueckgabe der verwendeten Funktion Text.Compare
-----------------------------------------------------------------------------*)
PROCEDURE Compare (self: T; otherObject:StorableObject.T): INTEGER =
  VAR adr: T;
  BEGIN
    adr := NARROW(otherObject,T);
    RETURN Text.Compare(self.name,adr.getName());
  END Compare;
(*-----------------------------------------------------------------------------
  Name:       ToText

  Aufgabe:    Die Methode fasst den Inhalt einer Adresse in einer Zeichenkette
              zusammen und gibt diese Zeichenkette zurueck. Die Funktion
              wird in AddressStorage verwendet, um eine Liste der Adressen
              damit erzeugen zu koennen.

  Parameter: keine

  Rueckgabe: die Adresse als ein Text
-----------------------------------------------------------------------------*)
PROCEDURE ToText (self: T):TEXT =
  BEGIN
    RETURN self.name & ", " & self.street & " " & self.num & ", "
           & self.post & " "& self.city;
  END ToText;
(*-----------------------------------------------------------------------------

  Die folgenden Methoden dienen dem Zugriff auf die internen Daten
  eines Address-Objekts. In diesem Beispiel sind die Methoden sehr einfach,
  sie lesen und schreiben jeweils genau ein Attribut des Objekts.

-----------------------------------------------------------------------------*)
PROCEDURE SetName (self: T; name: TEXT) =
  BEGIN
    self.name := name;
  END SetName;

PROCEDURE GetName (self: T):TEXT =
  BEGIN
    RETURN self.name
  END GetName;

PROCEDURE SetStreet(self: T; street:TEXT) =
  BEGIN
    self.street := street;
  END SetStreet;

PROCEDURE GetStreet(self: T):TEXT =
  BEGIN
    RETURN self.street;
  END GetStreet;

PROCEDURE SetNum(self: T; num:TEXT) =
  BEGIN
    self.num := num;
  END SetNum;

PROCEDURE GetNum(self: T):TEXT =
  BEGIN
    RETURN self.num;
  END GetNum;

PROCEDURE SetPost(self: T; post:TEXT) =
  BEGIN
    self.post := post;
  END SetPost;

PROCEDURE GetPost(self: T):TEXT =
  BEGIN
    RETURN self.post;
  END GetPost;

PROCEDURE SetCity(self: T; city:TEXT) =
  BEGIN
    self.city := city;
  END SetCity;

PROCEDURE GetCity(self: T):TEXT =
  BEGIN
    RETURN self.city;
```

```
  END GetCity;

BEGIN
END Address.



(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   ObjectStorage
  Datei:   ObjectStorage.i3
  Autor:   Dirk Jaeger

  ObjectStorage ist eine Weiterentwicklung von AddressStorage. Das Modul
  wurde zum einen als Klasse umformuliert und arbeitet jetzt auf Objekten
  der Klasse Storable Objekt. Es ist also nicht mehr auf das Speichern
  von Adressen eingeschraenkt. Adressen werden als Unterklassen von
  StorableObject definiert und koennen damit ebenfalls in diesem Modul
  abgelegt werden.
---------------------------------------------------------------------------*)
INTERFACE ObjectStorage;

IMPORT StorableObject;

EXCEPTION ObjectNotFound;

TYPE
  T <: Public;

  Public = OBJECT
  METHODS
    init    ():T ;
    insert  (newAdr:StorableObject.T) ;
    find    (proto:StorableObject.T): StorableObject.T RAISES {ObjectNotFound};
    delete  (proto:StorableObject.T) RAISES {ObjectNotFound};
    printAll();
END;

END ObjectStorage.

(*---------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   ObjectStorage
  Datei:   ObjectStorage.m3
  Autor:   Dirk Jaeger
---------------------------------------------------------------------------*)
MODULE ObjectStorage;

IMPORT SIO, StorableObject;

REVEAL
  T = Public BRANDED OBJECT

    head: REF Element;
    tail: REF Element;

  METHODS
    findElement(proto:StorableObject.T):REF Element := FindElement;
  OVERRIDES
    init    := Init;
    insert  := Insert;
    find    := Find;
    delete  := Delete;
    printAll:= PrintAll;
  END;

TYPE
  Element = RECORD     (* der Typ eines Listenelements *)
    adr:  StorableObject.T;
    next: REF Element;
    prev: REF Element;
  END;
```

```
(*-------------------------------------------------------------------------------
  Name:      Init

  Aufgabe:   Die Prozedur initialisiert den ADT, der durch eine doppelt
             verkettete Liste realisiert wird. Damit man beim Einfuegen
             und Loeschen nicht die Sonderfaelle (leere Liste, einfuegen am
             Ende, einfuegen am Anfang) behandeln muss, wird die Liste
             initial mit einem vordersten Element (head) und einem
             hintersten Element (tail) belegt. Diese beiden Element markieren
             den Anfang und das Ende der Liste. Sie enthalten keine Daten
             und werden nie geloescht. Neue Elemente werden deshalb immer
             zwischen zwei bereits vorhandenen Listenelementen eingefuegt,
             Sonderfaelle gibt es nicht.

  Parameter: adrst - eine Instanz des ADT

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE Init(self:T):T =
  BEGIN
    self.head := NEW (REF Element);
    self.tail := NEW (REF Element);
    self.head^.next := self.tail;
    self.head^.prev := NIL;
    self.head^.adr  := NEW(StorableObject.T);

    self.tail^.prev := self.head;
    self.tail^.next := NIL;
    self.tail^.adr  := NEW(StorableObject.T);

    RETURN self;
  END Init;
(*-------------------------------------------------------------------------------
  Name:      Insert

  Aufgabe:   fuegt eine neue Adresse in die Liste ein. Die Liste wird solange
             durchlaufen, wie das aktuellen Listenelements kleiner als
             das einzufuegenden Elements ist. Zum Vergleich der Elemente wird
             die Methode compare verwendet.
             Wenn diese Bedingung nicht mehr gilt, gibt es zwei Faelle:
             Entweder die beiden Elemente sind gleich, dann soll laut
             Aufgabenstelleung der alten Eintrag einfach durch den neuen
             ueberschrieben werden, oder die beiden Elemente sind nicht
             gleich, dann wird ein neues Element in die Liste eingefuegt.

  Parameter: adrst  - eine Instanz des ADT
             newAdr - die neue Adresse.

  Rueckgabe: keine
-------------------------------------------------------------------------------*)
PROCEDURE Insert(self:T; newAdr:StorableObject.T) =
  VAR
    newPointer, helpPointer : REF Element;

  BEGIN
    helpPointer:=self.head;

   (* Suche die Stelle zum Einfuegen in die Liste. *)

    WHILE helpPointer^.next^.adr.compare(newAdr) <= 0 AND
          helpPointer^.next # self.tail DO

      helpPointer:=helpPointer^.next;
    END;

    (* Pruefe ob die beiden Elemente gleich sind *)
    IF helpPointer^.adr.compare(newAdr) = 0 THEN

      helpPointer^.adr := newAdr;   (* alte Adresse wird ersetzt *)
    ELSE
      newPointer := NEW (REF Element);  (* neues Element anlegen und einfuegen *)
      newPointer^.adr         := newAdr;
      newPointer^.next        := helpPointer^.next;
      newPointer^.prev        := helpPointer;
      helpPointer^.next^.prev := newPointer;
      helpPointer^.next       := newPointer;
    END;
  END Insert;
```

```
(*-------------------------------------------------------------------------
  Name:       FindElement

  Aufgabe:    Die Prozedur sucht anhand eines Prototyps nach einem Listeneintrag.
              Die Liste wird solange durchlaufen, bis entweder der gesuchte
              Eintrag gefunden wurde, oder der aktuelle Eintrag
              groesser ist, als der gesuchte, oder das Ende der Liste erreicht
              wurde.

  Parameter: adrst - eine Instanz des ADT
              proto - eine Adresse, die denselben Schluessel hat, wie die
                      gesuchte Adresse

  Rueckgabe: einen Zeiger auf das gesuchte Element. Falls das gesuchte
              Element nicht in der Liste ist, wird NIL zurueckgegeben
-------------------------------------------------------------------------*)
PROCEDURE FindElement (self:T; proto:StorableObject.T): REF Element=
VAR  helpPointer : REF Element;

BEGIN
  helpPointer := self.head^.next;

  (* Suche nach einem Element mit demselben Schluessel *)
  WHILE (helpPointer^.adr.compare(proto) < 0) AND
    (helpPointer # self.tail) DO
    helpPointer:= helpPointer^.next;
  END;

  (* Wenn Listenende erreicht -> NIL *)
  IF helpPointer = self.tail THEN RETURN NIL;
  ELSE

    (* Wurde das Element gefunden? *)
    IF helpPointer^.adr.compare(proto) = 0 THEN

      RETURN helpPointer; (* Zeiger auf das gesuchte Element *)
    ELSE
      RETURN NIL;         (* sonst NIL zurueckgeben *)
    END;
  END;
END FindElement;
(*-------------------------------------------------------------------------
  Name:       Find

  Aufgabe:    Die Prozedur sucht nach einem Namen in der Liste und gibt
              die Adresse zu diesem Namen zurueck. Wenn der Name nicht
              in der Liste ist, wird eine Ausnahme erzeugt

  Parameter: adrst - eine Instanz des ADT
              proto - eine Adresse, die denselben Schluessel hat, wie die
                      gesuchte Adresse

  Rueckgabe: die Adresse zu dem gesuchten Namen
-------------------------------------------------------------------------*)
PROCEDURE Find(self:T; proto:StorableObject.T): StorableObject.T
  RAISES {ObjectNotFound} =
VAR pointer: REF Element;
BEGIN
  pointer := self.findElement(proto);
  IF pointer # NIL THEN
    RETURN pointer^.adr;
  ELSE
    RAISE ObjectNotFound;
  END;
END Find;
(*-------------------------------------------------------------------------
  Name:       Delete

  Aufgabe:    Die Prozedur loescht ein Element aus der Liste indem sie
              es aus der Listenstruktur auskettet.

  Parameter: adrst - eine Instanz des ADT
              proto - eine Adresse, die denselben Schluessel hat, wie die
                      gesuchte Adresse
  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE Delete (self:T; proto:StorableObject.T)
  RAISES {ObjectNotFound}=
VAR pointer: REF Element;
  BEGIN
    pointer := self.findElement(proto);
    IF pointer # NIL THEN
      pointer^.prev^.next := pointer^.next;
      pointer^.next^.prev := pointer^.prev;
      pointer := NIL;
    ELSE
      RAISE ObjectNotFound;
    END;
  END Delete;
```

```
(*-------------------------------------------------------------------------
  Name:      PrintAll

  Aufgabe:   gibt die gesamte Liste aus

  Parameter: adrst - eine Instanz des ADT

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintAll(self:T) =
VAR helpPointer: REF Element;
  BEGIN
    helpPointer := self.head;
    WHILE helpPointer # NIL DO
      SIO.PutText( helpPointer^.adr.toText());
      SIO.Nl();
      helpPointer:= helpPointer^.next;
    END;
  END PrintAll;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN
END ObjectStorage.
```

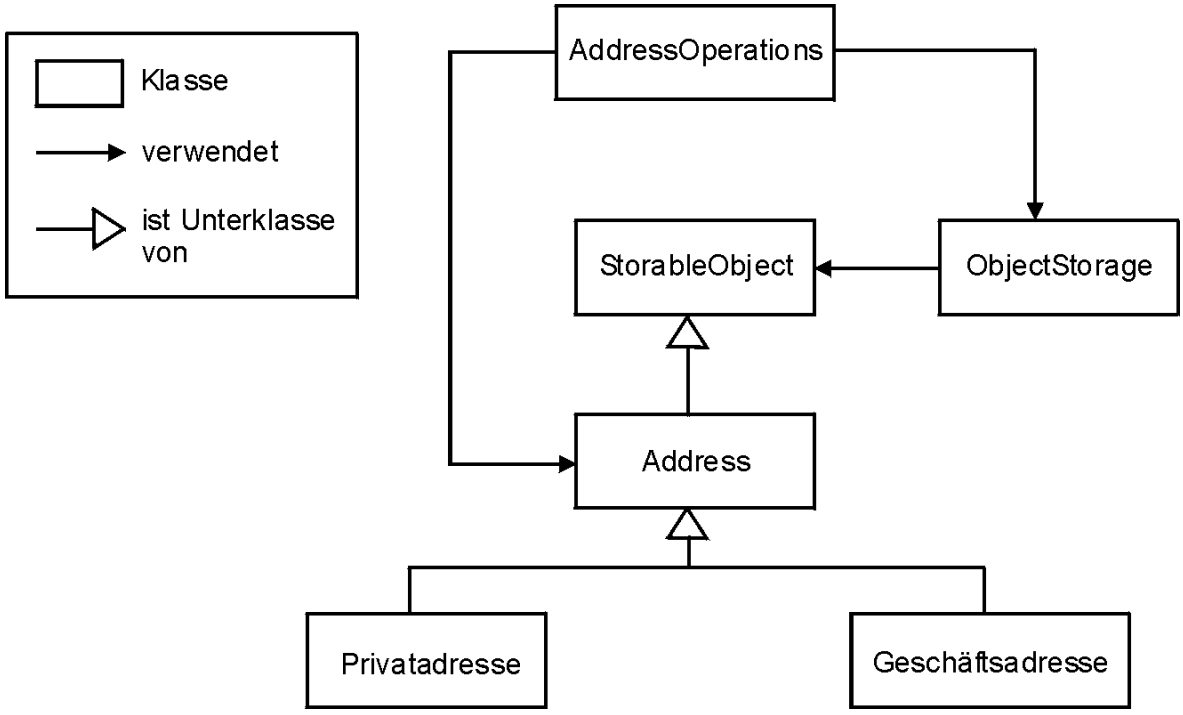
```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   AddressOperations
  Datei:   AddressOperations.i3
  Autor:   Dirk Jaeger
-------------------------------------------------------------------------*)
INTERFACE AddressOperations;

IMPORT SIO;

PROCEDURE PrintList() ;
PROCEDURE InputAddress() RAISES {SIO.Error};
PROCEDURE SearchAddress() RAISES {SIO.Error};
PROCEDURE RemoveName() RAISES {SIO.Error};

END AddressOperations.
```

```
(*-------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   AddressOperations
  Datei:   AddressOperations.m3
  Autor:   Dirk Jaeger

  AddressOperations stuetzt sich in dieser Version auf die Klassen
  ObjectStorage und Address ab. Es legt ein Objekt der Klasse
  ObjectStorage an und verwendet es, um darin Objekte der Klasse
  Address zu verwalten.
-------------------------------------------------------------------------*)
MODULE AddressOperations;

IMPORT Address;
IMPORT ObjectStorage;
IMPORT SIO;

VAR Adrst:ObjectStorage.T;  (* zentraler Datenspeicher des ADO *)

(*-------------------------------------------------------------------------
  Name:      PrintList

  Aufgabe:   gibt die gesamte Liste aus

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE PrintList() =
  BEGIN
    Adrst.printAll();
  END PrintList;
```

```
(*-------------------------------------------------------------------------
  Name:       InputAddress

  Aufgabe:    liest eine Adresse vom Benutzer ein und fuegt sie in
              die Liste ein.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE InputAddress() RAISES {SIO.Error}=
VAR newAdr: Address.T;
  BEGIN
    newAdr := NEW(Address.T);
    SIO.PutText ("Name         : ");
    newAdr.setName(SIO.GetLine());
    SIO.PutText ("Strasse      : ");
    newAdr.setStreet(SIO.GetLine());
    SIO.PutText ("Hausnummer   : ");
    newAdr.setNum(SIO.GetLine());
    SIO.PutText ("Postleitzahl: ");
    newAdr.setPost(SIO.GetLine());
    SIO.PutText ("Ort          : ");
    newAdr.setCity(SIO.GetLine());

    Adrst.insert(newAdr);
  END InputAddress;
(*-------------------------------------------------------------------------
  Name:       SearchAddress

  Aufgabe:    liest einen Namen vom Benutzer ein und ruft mit diesem Namen
              der Prozedur Find, auf um danach zu suchen.

  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE SearchAddress()  RAISES {SIO.Error} =
 VAR name:TEXT;
     proto,adr:Address.T;

 BEGIN
   SIO.PutText("Nach welchem Namen soll gesucht werden: ");
   name:=SIO.GetLine();
   proto:=NEW(Address.T);
   proto.setName(name);
   TRY
     adr := Adrst.find(proto);
     SIO.PutText ("Die Adresse lautet: \n\n");
     SIO.PutText (adr.getName()); SIO.Nl();
     SIO.PutText (adr.getStreet() & " " & adr.getNum());  SIO.Nl();
     SIO.PutText (adr.getPost()   & " " & adr.getCity()); SIO.Nl();
     SIO.Nl();
   EXCEPT
   | ObjectStorage.ObjectNotFound =>
     SIO.PutText ("\nDieser Name ist nicht in der Liste!\n")
   END;
 END SearchAddress;

(*-------------------------------------------------------------------------
  Name:       RemoveName

  Aufgabe:    liest einen Namen vom Benutzer ein und loescht den Eintrag
              mit diesem Namen falls vorhanden.
  Parameter: keine

  Rueckgabe: keine
-------------------------------------------------------------------------*)
PROCEDURE RemoveName()  RAISES {SIO.Error} =
 VAR name:TEXT;
     proto:Address.T;
 BEGIN
   SIO.PutText("Welcher Namen soll geloescht werden: ");
   name:=SIO.GetLine();
   proto := NEW(Address.T);
   proto.setName(name);
   TRY
     Adrst.delete(proto);
     SIO.PutText ("\nDer Eintrag wurde geloescht!\n")
   EXCEPT
   | ObjectStorage.ObjectNotFound =>
     SIO.PutText ("\nDieser Name ist nicht in der Liste!\n");
   END;
 END RemoveName;
(*-------------------------------------------------------------------------
  Hauptprogramm
-------------------------------------------------------------------------*)
BEGIN

  Adrst:=NEW(ObjectStorage.T).init();  (* Initialisiere den Datenspeicher *)
END AddressOperations.
```

```
(*-----------------------------------------------------------------------------
  Lehrstuhl fuer Informatik III
  Prof. Dr.-Ing. M. Nagl

  Vorlesung Programmierung - WS 1999/2000
  Loesung der Uebungsaufgabe 13.2

  Modul:   AddressMain
  Datei:   AddressMain.m3
  Autor:   Dirk Jaeger
-----------------------------------------------------------------------------*)
MODULE AddressMain EXPORTS Main;

IMPORT SIO, Text;
FROM AddressOperations IMPORT InputAddress, PrintList, SearchAddress, RemoveName;

VAR input:TEXT;
(*-----------------------------------------------------------------------------
  Hauptprogramm
-----------------------------------------------------------------------------*)
BEGIN
  TRY
    REPEAT
      SIO.PutText("\n---------------------------------\n");
      SIO.PutText("      M E N U E          \n");
      SIO.PutText("                         \n");
      SIO.PutText(" 1 - Neue Adresse eingeben   \n");
      SIO.PutText(" 2 - Anzeigen der Liste      \n");
      SIO.PutText(" 3 - Suchen nach einem Namen \n");
      SIO.PutText(" 4 - Loeschen eines Elements  \n");
      SIO.PutText(" 5 - Beenden des Programms   \n");
      SIO.PutText("\n\n");
      SIO.PutText("Bitte waehlen Sie eine Funktion: ");
      input:=SIO.GetLine() & " ";
      SIO.Nl();
      SIO.PutText("\n---------------------------------\n");
      SIO.Nl();

      CASE Text.GetChar(input,0) OF
      | '1' => InputAddress();
      | '2' => PrintList();
      | '3' => SearchAddress ();
      | '4' => RemoveName ();
      ELSE END;

    UNTIL Text.GetChar(input,0) ='5';
  EXCEPT SIO.Error => SIO.PutText("*** Fehler im Modul SIO ***\n");
  END;
END AddressMain.
```

b)



Aufgabe 13.3

```
%-------------------------------------------------------------------------------
%  Lehrstuhl fuer Informatik III
%  Prof. Dr.-Ing. M. Nagl
%
%  Vorlesung Programmierung - WS 1999/2000
%  Loesung der Uebungsaufgabe 13.3
%
%  Datei:   relative.pl
%  Autor:   Dirk Jaeger
%-------------------------------------------------------------------------------*)

%
% zuerst definieren wir, wer wessen Ehefrau ist.
%

ehefrau('Helga','Kurt').
ehefrau('Sabine','Frank').
ehefrau('Karin','Andreas').
ehefrau('Maike','Olaf').
ehefrau('Annette','Peter').
ehefrau('Maria','Stefan').

%
% X ist Ehemann von Y, wenn Y Ehefrau von X ist.
%

ehemann(X,Y) :- ehefrau(Y,X).

%
% Und verheiratet sind X und Y wenn X entweder Ehemann
% oder aber Ehefrau von Y ist.
%

verheiratet(X,Y) :- ehefrau(X,Y).
verheiratet(X,Y) :- ehemann(X,Y).

%
% Als naechstes definieren wir, wer wessen Vater ist
%

vater('Kurt','Horst').
vater('Kurt','Stefan').
vater('Stefan','Holger').
vater('Stefan','Peter').
vater('Peter','Anne').
vater('Peter','Werner').
vater('Peter','Ruth').
vater('Frank','Maria').
vater('Frank','Anja').
vater('Frank','Thomas').
vater('Frank','Andreas').
vater('Andreas','Tobias').
vater('Andreas','Stefanie').
vater('Andreas','Olaf').
vater('Olaf','Florian').
vater('Olaf','Sarah').

%
% Die Mutter einer Person, ist die Person, die mit
% dem Vater verheiratet ist: Diese Regel gilt natuerlich
% nur fuer das Beispiel und nicht in der Realitaet, da
% Scheidungen und uneheliche Kinder nicht beruecksichtigt
% sind. Will man diese Faelle ebenfalls abdecken, muss
% man auch die Mutter als Faktum definieren.
%

mutter(X,Y) :- vater(Z,Y), verheiratet(X,Z).

%
% Ausserdem muessen wir noch das Geschlecht der
% beteiligten Personen festlegen: Weiblich sind:
%

weiblich('Helga').
weiblich('Sabine').
weiblich('Maria').
weiblich('Anja').
weiblich('Karin').
weiblich('Annette').
weiblich('Stefanie').
weiblich('Maike').
weiblich('Anne').
weiblich('Ruth').
weiblich('Sarah').

%
% Maennlich sind alle die, die nicht weiblich sind.
%

maennlich(X) :- not (weiblich(X)).
```

```
%
% Die Definition von Elternteil vereinfacht spaeter die
% Definition von Onkel, Tante, Grossvater und Grossmutter
%

elternteil(X,Y) :- vater(X,Y).
elternteil(X,Y) :- mutter(X,Y).

%
% X ist Kind von Y wenn Y entweder dessen Mutter oder Vater ist
%

kind(X,Y) :- mutter(Y,X).
kind(X,Y) :- vater(Y,X).

%
% Toechter sind die weiblichen Kinder, Soehne die maennlichen
%

tochter(X,Y) :- kind(X,Y), weiblich(X).
sohn(X,Y)    :- kind(X,Y), maennlich(X).

%
% Ein Bruder einer Person X ist jemand, der den gleichen vater
% hat, aber nicht die Person X selber ist und maennlich ist.
% Analog fuer die Schwester, mit dem Unterschied, dass sie
% weiblich ist.
%

bruder(X,Y)    :- vater(Z,X), vater(Z,Y), not(X=Y), maennlich(X).
schwester(X,Y) :- vater(Z,X), vater(Z,Y), not(X=Y), weiblich(X).

%
% Ein Grossvater ist der Vater eines Elternteils,
% eine Grossmutter ist die Mutter eines Elternteils.
%

grossvater(X,Y)  :- elternteil(Z,Y),vater(X,Z).
grossmutter(X,Y) :- elternteil(Z,Y), mutter(X,Z).

%
% Onkel und Tanten sind Brueder bzw. Schwestern von
% Elternteilen und die Ehepartner von Onkeln und
% Tanten.
%

tante(X,Y) :- elternteil(Z,Y), schwester(X,Z).
tante(X,Y) :- ehefrau(X,Z),bruder(Z,W),elternteil(W,Y).
onkel(X,Y) :- elternteil(Z,Y), bruder(X,Z).
onkel(X,Y) :- ehemann(X,Z),schwester(Z,W),elternteil(W,Y).

%
% Ein Cousin ist der Sohn eines Onkels oder einer Tante.
% Eine Cousine ist die Tocher eines Onkels oder einer Tante.
%

cousin(X,Y) :- tante(Z,Y), sohn(X,Z).
cousin(X,Y) :- onkel(Z,Y), sohn(X,Z).
cousine(X,Y) :- tante(Z,Y), tochter(X,Z).
cousine(X,Y) :- onkel(Z,Y), tochter(X,Z).
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **14. Übungsblatt**

Musterlösung

Aufgabe 14.1

```
gewinnsituation(AnzahlHölzer):-integer(AnzahlHölzer),AnzahlHölzer > 0,  (*)
                               (AnzahlHölzer mod 4) =\= 0.               (**)
```

`AnzahlHölzer` muß eine positive Integer-Zahl sein (ausgedrückt durch **(*)**).

Es liegt immer eine Gewinnsituation vor, wenn der am Zug befindliche Spieler eine Anzahl Streichhölzer vorfindet, die **kein** Vielfaches von 4 ist.

Wenn 4 Streichhölzer daliegen, kann der am Zug befindliche Spieler 1 bis 3 wegnehmen. Es bleibt auf jeden Fall eine Anzahl Streichhölzer übrig, die sein Gegenspieler in einem Zug abräumen kann. Analog gilt dies für jedes Vielfache von Vier (ausgedrückt durch **(**)**).

Aufgabe 14.2

a)

(CAR (QUOTE (X Y Z)))
alternative Schreibweise (CAR '(X Y Z))

CHAR liefert das erste Element einer Liste

=> X

b)

(PLUS 7 (CAR (QUOTE(2 8))))
alternative Schreibweise (+ 7 CAR '(2 8))

=> (Auswertung von CHAR liefert) (+ 7 2)
=> (Auswertung von +)
=> 9

c)

(CDR (LIST (QUOTE (4 6)) 3))
alternative Schreibweise (CDR (LIST '(4 6) 3))

=> (Auswertung von LIST liefert) (CDR '('(4 6) 3))
; LIST macht aus seinen einzelnen Argumenten eine Liste
=> (Auswertung von CDR)
; erstes Listenelement ist die zweielementige Liste (4 6)
=> (3)
; CDR liefert als Restliste die einelementige Liste (3) und nicht das Atom 3 !

d)

(CONS 3 (LIST 7 8 1))

=> (Auswertung von LIST liefert) (CONS 3 '(7 8 1)
; List macht eine Liste aus beliebig vielen Argumenten
=> (Auswertung von CONS)
; CONS bekommt zwei Argumente, davon ist zweite hier eine Liste, und fügt das erste
; Argument vorn in diese Liste ein
=> (3 7 8 1)

e)

(CONS (QUOTE CDR) (LIST (QUOTE P)))
alternative Schreibweise (CONS 'CDR (LIST '(P)))

=> (Auswertung von LIST liefert) (CONS 'CDR '(P))
; LIST erzeugt eine einelementige Liste
=> (CDR P)
; CONS hängt den String „CDR“ vor das Listenelement „P“ in eine
; gemeinsame Liste ein

Aufgabe 14.3
a)

```
(DEFUN DIVIDEBACK (X Y)
        (REVERSE (DIVIDE (X Y))
)
```

b)

```
(DEFUN FIBON (N)
            ( COND      ((= N 1) 1)                          ; N = 1
                        ((= N 2) 1)                          ; N = 2
                        (T ( + FIBON(N-1)) (FIBON(N-2)))     ; ELSE-Teil
            )
)
```

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
{*katja|jaeger*}*@i3.informatik.rwth-aachen.de*

Programmierung WS 1999/2000 1. Übungsblatt

Ausgabe: Montag, 18.10.99 **Abgabe bis:** Montag, 25.10.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 28. und 29.10.99

Aufgabe 1.1 ***(5 Punkte)***
Formulieren Sie in strukturierter Umgangssprache einen Algorithmus, der beschreibt, wie Sie eine Telefonnummer bei der Auskunft der Telekom nachfragen. Berücksichtigen Sie dabei auch Sonderfälle. (Handelt es sich um eine Nummer im Inland oder im Ausland?) Gehen Sie davon aus, daß Ihnen die Nummer der Auskunft nicht bekannt ist und Sie diese im Telefonbuch nachschlagen müssen. Beginnen Sie mit diesem Schritt des Algorithmus.

Aufgabe 1.2 ***(5 Punkte)***
Gegeben sei ein einfacher Von-Neumann-Rechner mit 4 Registern, der die folgenden Befehle ausführen kann:

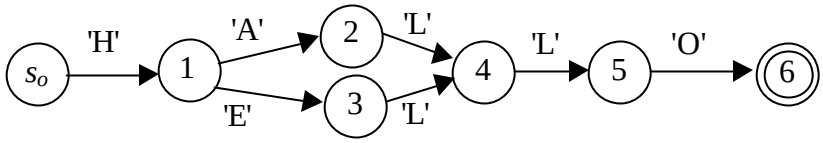
Befehl		Bedeutung
`LoadAdr`	*n, x*	Lade den Inhalt der Speicheradresse *x* in das Register *n*.
`LoadVal`	*n, j*	Lade den Wert *j* in das Register *n*.
`StoreAdr`	*n, x*	Schreibe den Inhalt des Registers *n* in die Speicheradresse *x*.
`Add`	*n, m*	Addiere den Inhalt des Registers *m* zum Inhalt des Registers *n*. Das Ergebnis steht danach im Register *n*.
`Sub`	*n, m*	Subtrahiere den Inhalt des Registers *m* vom Inhalt des Registers *n*. Das Ergebnis steht danach in Register *n*.
`Dec`	*n*	Vermindere den Inhalt des Registers *n* um eins.
`Inc`	*n*	Erhöhe den Inhalt des Registers *n* um eins.
`Jmp`	*x*	Setze die Ausführung des Programms mit dem Befehl fort, der an der Speicheradresse *x* steht.
`Compare`	*n, m*	Vergleiche den Inhalt des Registers *n* mit dem Inhalt des Registers *m*.
`JmpEqual`	*x*	Wenn der zuletzt ausgeführte Vergleich `Compare` *n, m* ergeben hat, daß der Inhalt von Register *n* genauso groß war als der Inhalt von Register *m*, dann setze die Ausführung des Programms mit dem Befehl fort, der an der Speicheradresse *x* steht.
`JmpGreater`	*x*	Wenn der zuletzt ausgeführte Vergleich `Compare` *n, m* ergeben hat, daß der Inhalt von Register *n* größer war als der Inhalt von Register *m*, dann setze die Ausführung des Programms mit dem Befehl fort, der an der Speicheradresse *x* steht.
`JmpLesser`	*x*	Wenn der zuletzt ausgeführte Vergleich `Compare` *n, m* ergeben hat, daß der Inhalt von Register *n* kleiner war als der Inhalt von Register *m*, dann setze die Ausführung des Programms mit dem Befehl fort, der an der Speicheradresse *x* steht.

Schreiben Sie ein Programm, daß eine Zahl, die in der Speicheradresse 20 abgelegt ist, quadriert und das Ergebnis an der Speicheradresse 21 ablegt. Das Programm soll im Speicher ab der Adresse 1 liegen. Jeder Befehl belegt genau eine Speicheradresse. Alle Zahlen werden im Dezimalsystem angegeben. Schreiben Sie das Programm in Form einer Liste von Befehlen und schreiben Sie vor jeden Befehl die Speicheradresse, an der dieser Befehl abgelegt ist. Kommentieren Sie, begleitend zu den Befehlen, Ihr Programm.

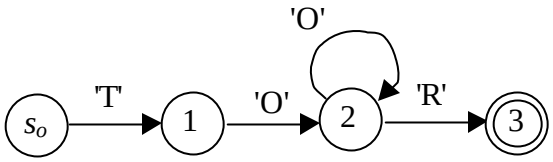
Aufgabe 1.3 ***(5 Punkte)***

Ein *endlicher Automat* ist ein mathematisches Modell für einen einfachen Algorithmus, der z.B. Zeichenfolgen liest und diese entweder *akzeptiert* oder *verwirft*. Das Verhalten eines endlichen Automaten läßt sich durch ein Zustandsübergangsdiagramm beschreiben. *Zustände* werden als Kreise, *Zustandsübergänge* als Pfeile dargestellt. Den Zustand, in dem sich der Automat am Anfang befindet, nennt man den *Startzustand* 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 nach dem Lesen des letzten Zeichens in einem Endzustand ist.

Beispiel: Der folgende Automat akzeptiert genau die beiden Zeichenfolgen "HALLO" und "HELLO":



Es ist erlaubt, daß der Automat durch einen Zustandsübergang wieder in einen Zustand gelangt, in dem er vorher war. Der folgende Automat akzeptiert die Zeichenfolgen "TOR", "TOOR", "TOOOR", "TOOOOR", usw.



Erstellen Sie das Zustandsübergangsdiagramm eines endlichen Automaten, der als Eingabe Zeichenketten akzeptiert, die Zahlen in der vom Taschenrechner bekannten E-Notation bilden.

Beispiel: "1.0E01", "-1.012321E05", "2.1321E-22", "8.0099E-03"

Eine Zahl in der E-Notation kann ein Minus als Vorzeichen haben, danach folgt eine Stelle vor dem Dezimalpunkt, dann der Dezimalpunkt und danach mindestens eine Stelle hinter dem Dezimalpunkt. Am Ende der Zahl steht das Zeichen 'E' gefolgt von einem Exponenten. Der Exponent hat genau zwei Ziffern, vor denen ein Minus als Vorzeichen stehen kann.

Aufgabe 1.4 ***(5 Punkte)***

Verschiedene Zahlensysteme und ihre Darstellung im Rechner spielen eine wichtige Rolle in der Informatik. Zahlen werden heute vorrangig nach dem Prinzip des Dezimalsystems, einem *Positionensystem* dargestellt. Dieses 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.

Übertragen Sie die folgenden Zahlen aus der gegebenen Dezimaldarstellung jeweils in das geforderte Zahlensystem. Übertragen Sie

72329	und	18.453125	in das Binärsystem (Basis 2)
129700	und	325.060546875	in das Oktalsystem (Basis 8)
2729309	und	31300424.05078125	in das Hexadezimalsystem (Basis 16)

Hinweis: Im Hexadezimalsystem werden die Buchstaben 'A' bis 'F' als zusätzliche Ziffern verwendet (A=10, B=11, ..., F=15).

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung WS 1999/2000 2. Übungsblatt

Ausgabe: Montag, 25.10.99 **Abgabe bis:** Dienstag, 2.11.99, 13.00 Uhr

Besprechung: in den Übungsgruppen am 4. und 5.11.99

Aufgabe 2.1 ***(6 Punkte)***

a) Gegeben sei folgende Grammatik G = ({A, B, C, D}, {a, b, c}, P, S} mit

P = {
S -> A
A -> Ba
B -> aA
A -> Cb
C -> bA
A -> DAD
D -> c
A -> ε
A -> a
A -> b
A -> c
}

Beschreiben Sie informell, welche Sprache durch diese Grammatik definiert wird. Wählen Sie ein beliebiges Wort der Länge 8 aus der Sprache aus und geben Sie dafür den Ableitungsbaum an. *(2 Punkte)*

b) Geben Sie eine Grammatik an, die eine Sprache mit den Worten der Form $a^n b^n$ (n>0) definiert, Beispielwort der Sprache: aaaabbbb, n=4. *(2 Punkte)*

c) Geben Sie eine Grammatik an, die eine Sprache definiert, deren Worte aus Folgen von Ziffern bestehen, die durch ein Plus- oder Minus-Zeichen voneinander getrennt sind, Beispielwort der Sprache: 9+5-6-2+2. *(2 Punkte)*

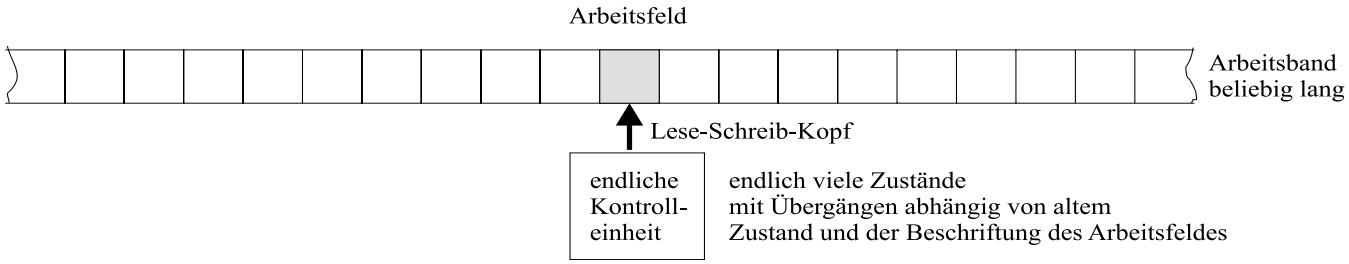
Aufgabe 2.2 ***(7 Punkte)***

a) Definieren Sie die Syntax der EBNF mit Hilfe von Syntaxdiagrammen. *(3 Punkte)*

b) Definieren Sie die Syntax der EBNF in EBNF-Notation. *(4 Punkte)*

Aufgabe 2.3 ***(7 Punkte)***

Im Jahre 1936 hat A.M. Turing eine ausführbares Modell für die formale Berechnung von Algorithmen eingeführt – die Turing-Maschine, die das Rechnen mit Bleistift und Papier nachbildet. Auf einem Band, das als Eingabe und als Arbeitsspeicher dient, stehen verschiedene Zeichen. Die Maschine hat eine endliche Kontrolleinheit, d.h. kann endlich viele Zustände annehmen.



Als Aktionen stehen zur Verfügung:

- Lesen des Zeichens auf dem Arbeitsfeld
- Schreiben eines Zeichens auf das Arbeitsfeld
- Bewegen des LS-Kopfes um eine Position nach links oder nach rechts

Alle drei Aktionen werden in einem Arbeitsschritt durchgeführt.

Was die TM in Abhängigkeit von dem Zustand der Kontrolleinheit und einem gelesenen Symbol tut, wird z.B. durch eine Tabelle festgelegt. Unter den Zuständen gibt es einen ausgezeichneten Anfangs- und eine Menge von Endzuständen, für die kein weiterer Übergang definiert ist.

Beispiel:
Die Maschine ist im Zustand *Zs* , hat ein Zeichen 0 gelesen, geht in den gleichen Zustand über, schreibt ein Zeichen Blank (B) auf das Arbeitsfeld und bewegt sich zum nächsten rechten Zeichenfeld.

Zustand	Eingabe	Zustand'	neues Zeichen	{L,R,N}
Z_s	0	Z_s	B	R

L=links, R=rechts, N=neutral

Wir verwenden eine Bandbeschriftung mit Strichdarstellung für Zahlen, d.h. so viele Striche wie der Wert der Zahl beträgt. Auf dem Band befinde sich #*s(a)*Bs*(b)*#, wobei s(a) die Strichdarstellung von *a* ist, entsprechend *s(b)* die von *b*. Die # sind Randsymbole, die beiden Eingaben sind durch ein Leerzeichen B voneinander getrennt.

Wie sieht die Turingtafel aus, die als Ergebnis #*s(a+b)*# liefert? Die Maschine sei nach Berechnung dieses Wertes im Zustand Z_c und befinde sich auf dem ersten Strich von *s(a+b)*.

Geben Sie die Tafel in Gruppen von Zustandsübergängen an und geben Sie in einem Kommentar an, was die Gruppen jeweils machen.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung WS 1999/2000 3. Übungsblatt

Ausgabe: Dienstag, 2.11.99 **Abgabe bis:** Montag, 8.11.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 11. und 12.11.99

Hinweis zur Lösung der Programmieraufgaben 3.2 bis 3.4: Die als Lösung für dieses Übungsblatt zu erstellenden Programme sollen rein funktional programmiert werden. D.h. es dürfen nur die Konstrukte von Modula-3 verwendet werden, die in der Vorlesung im Abschnitt zur Funktionalen Programmierung vorgestellt wurden. Dabei handelt es sich im wesentlichen um Funktionsaufrufe sowie die bedingte Ausführung von Anweisungen mit `IF-THEN-ELSE`- und `CASE`-Ausdrücken. Insbesondere ist die Verwendung von Variablen, Konstanten und Schleifen untersagt. Lösungen, die sich nicht an diese Richtlinien halten, werden nicht akzeptiert.
Für Ein- und Ausgaben verwenden Sie die Funktionen der Bibliothek `SIO`. Für Operationen auf Texten verwenden Sie die Funktionen der Bibliothek `Text`.

Aufgabe 3.1 ***(4 Punkte)***
Geben Sie eine Grammatik an, mit der man korrekte E-Mail-Header erzeugen kann. Folgende informelle Beschreibung charakterisiert den Aufbau der Header:

TO:	Im TO-Feld steht die Adresse des Empfängers. Es können auch mehrere Empfänger angegeben sein, wobei die Adressen dann durch ein Komma voneinander getrennt sind. Prinzipiell gibt es zwei Möglichkeiten, wie die Adresse angegeben sein kann. Entweder steht im TO-Feld nur die E-Mail-Adresse in der Form name@domain, oder sie steht zusammen mit einem voranstehenden Kommentar (z.B. dem Namen des Empfängers). In letzterem Fall steht die E-Mail-Adresse in spitzen Klammern. Beispiel: Vorname Nachname <name@domain>
CC:	Im CC-Feld (Carbon Copy = Durchschlag bzw. Kopie) stehen die Adressen, die eine Kopie der Mail bekommen. Die Nachricht ist also nicht direkt an diese Personen gerichtet, aber sie sollen quasi "zur Kenntnisnahme" die Nachricht erhalten. Die Syntax ist wie im TO-Feld.
BCC:	Das BCC-Feld (Blind Carbon Copy) entspricht dem CC-Feld, nur daß dieses Feld und damit auch die dort aufgeführten Adressen für den Empfänger unsichtbar sind. Die unter TO: und CC: aufgeführten Empfänger erfahren also nicht, daß die unter BCC: aufgeführten Personen die Nachricht auch bekommen haben. Die Syntax ist wie im TO-Feld.
SUBJECT:	Das SUBJECT-Feld enthält einen aussagekräftigen Text, der den Inhalt der Nachricht beschreibt.
DATE:	Das DATE-Feld gibt an, wann die Nachricht verschickt wurde. Die Syntax des Datums ist TT:MM:JJ.
FROM:	Im FROM-Feld steht die Adresse des Absenders. Die Syntax ist wieder wie im TO-Feld.

Die Feldbezeichner sind in Großbuchstaben anzugeben, gefolgt von einem ":" und einem Leerzeichen. Die Reihenfolge der Felder soll wie oben angegeben sein.

Aufgabe 3.2 ***(4 Punkte)***
Schreiben Sie ein rein funktionales, rekursives Programm, das die Worte eines Textes einzeln vom Benutzer einliest und anschließend in umgekehrter Reihenfolge wieder ausgibt. Die Reihenfolge der Buchstaben innerhalb der Worte soll erhalten bleiben. Das Programm soll solange weitere Worte einlesen, bis der Benutzer die Zeichenfolge "\STOP" eingibt. Danach sollen die Worte hintereinander ohne Zeilenumbruch und jeweils durch ein Leerzeichen getrennt ausgegeben werden. Das abschließende Wort "\STOP" soll nicht ausgegeben werden.

Aufgabe 3.3 ***(5 Punkte)***
Ein Term der Form $(x + y)^n$ ergibt ausmultipliziert den Ausdruck:

$$(x+y)^n = \sum_{i=0}^{n} a_i x^{n-i} y^i = a_0 x^n + a_1 x^{n-1} y + a_2 x^{n-2} y^2 + \ldots + a_{n-1} x y^{n-1} + a_n y^n$$

(Anmerkung: für n=2 erhält man die bekannte 1. Binomische Formel: $(x+y)^2 = x^2 + 2xy + y^2$.)
Die Koeffizienten a_i für eine gegebenes n lassen sich auf einfache Weise über ein Zahlenschema bestimmen, das sogenannte *Pascalsche Dreieck.*

n																					
0											1										
1									1			1									
2								1			2			1							
3							1			3			3			1					
4						1			4			6			4			1			
5					1			5			10			10			5			1	
6				1			6			15			20			15			6			1
...			1			...			...			...			...			...			...			1

Das Schema beginnt mit der Zeile 0. Die Zeile n enthält jeweils die Koeffizienten von a_0 bis a_n für diesen Wert von n. Der erste und der letze Wert jeder Zeile ist eine Eins. Die anderen Werte der Zeile n berechnen sich, indem man die beiden Werte, die im Dreieck in der Zeile n-1 links und rechts darüber stehen addiert. Als Beispiel ist in der Abbildung oben dargestellt, wie sich der Wert a_4 für n=6 aus den darüberliegenden Werten a_3 und a_4 der Zeile n=5 berechnet.

a) Schreiben Sie ein rein funktionales, rekursives Programm in Modula-3, das die Werte n und i vom Benutzer als Eingabe entgegennimmt und den Koeffizienten a_i für die Zeile n ausgibt. Für den Fall, daß das eingegebene i größer ist als n, soll das Programm den Wert null zurückgeben. *(3 Punkte)*

b) Geben Sie die zeitliche Abfolge und Verschachtelung der Funktionsaufrufe und Rücksprünge ihres Programms für die Eingaben n=5 und i=2 in Form eine Liste an. Geben Sie zu jedem Funktionsaufruf die aktuellen Parameter und zu jedem Rücksprung den berechneten Funktionswert an. Machen Sie die verschiedenen Rekursionstiefen durch Einrückungen des Textes kenntlich.
(2 Punkte)

Aufgabe 3.4 ***(7 Punkte)***
Schreiben Sie ein rein funktionales, rekursives Programm, daß einen Text zeilenweise einliest. Es sollen solange Textzeilen eingelesen werden, bis der Benutzer die Zeile "\STOP" eingibt. Das Programm soll im eingegebenen Text alle Vokale durch Unterstriche "_" ersetzen und den derart geänderten Text wieder ausgeben. Die Zeilenaufteilung des Textes soll dabei erhalten werden, d.h. die Worte, die der Benutzer als ein Zeile eingegeben hat, sollen auch als eine Zeile wieder ausgegeben werden. Das abschließende Wort "\STOP" soll nicht ausgegeben werden.

Testen Sie Ihr Programm mit dem Aufgabentext der Aufgabe 3.2 als Eingabe. Geben Sie zusammen mit dem Programmtext auch einen Ausdruck ab, der die Ausgabe Ihres Programms für dieses Beispiel dokumentiert.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **4. Übungsblatt**

Ausgabe: Montag, 8.11.99 **Abgabe bis:** Montag, 15.11.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 18. und 19.11.99

Hinweis zur Lösung der Programmieraufgaben 4.2 und 4.3: Die als Lösung für dieses Übungsblatt zu erstellenden Programme sollen rein funktional programmiert werden. D.h. es dürfen nur die Konstrukte von Modula-3 verwendet werden, die in der Vorlesung im Abschnitt zur Funktionalen Programmierung vorgestellt wurden. Dabei handelt es sich im wesentlichen um Funktionsaufrufe sowie die bedingte Ausführung von Anweisungen mit `IF-THEN-ELSE`- und `CASE`-Ausdrücken. Insbesondere ist die Verwendung von Variablen, Konstanten und Schleifen untersagt. Lösungen, die sich nicht an diese Richtlinien halten, werden nicht akzeptiert.
Für Ein- und Ausgaben verwenden Sie die Funktionen der Bibliothek `SIO`. Für Operationen auf Texten verwenden Sie die Funktionen der Bibliothek `Text`.

Aufgabe 4.1 ***(5 Punkte)***

a) In Modula-3 gibt es eine Reihe von kontextfreien Syntaxangaben, die dennoch wie die kontextsensitiven, d.h. umgangssprachlich, festgelegt worden sind. Hierzu gehört etwa, daß der Bezeichner einer Funktion, wenn er am Ende wiederholt wird, der gleiche Bezeichner sein muß. Beschreiben Sie diesen Sachverhalt über EBNFs, wobei angenommen wird, daß die Bezeichner eine Länge von 3 Zeichen haben. *(2 Punkte)*

b) Folgende EBNF ist wegen ihrer Strukturierung unleserlich. Beschreiben Sie die definierte Sprache. Geben Sie eine sauber strukturierte und gut lesbare Fassung an, so daß die definierte Sprache mit der angegebenen EBNF übereinstimmt. Terminalsymbole stehen in Hochkommata. (*3 Punkte)*

```
D_NUMERAL   = DSEQ UL_DSEQ .
UL_DSEQ     = "_"D_NUMERAL | D_NUMERAL | ε .
DSEQ        = D DSEQ | D | ε .
D           = ODD_D | EVEN_D .
ODD_D       = "1" | "3" | "5" | "7" | "9"  .
EVEN_D      = "0" | "2" | "4" | "6" | "8" .
```

Aufgabe 4.2 ***(5 Punkte)***

Schreiben Sie ein rein funktionales, rekursives Programm, das vom Benutzer ein Wort als Eingabe entgegennimmt und entscheidet, ob das eingegebene Wort ein Palindrom ist.

a) Erstellen Sie zunächst ein Funktionsnetz, das die Arbeitsweise dieses Programms beschreibt.

b) Erstellen Sie das Programm in Modula-3. Das Programm soll eine Aufforderung zur Eingabe an den Benutzer richten. Falls das eingegebene Wort ein Palindrom ist, soll folgender Texte ausgegeben werden: "Das Wort <*eingabe*> ist ein Palindrom". Falls das Wort kein Palindrom ist, soll ausgegeben werden: "Das Wort <*eingabe*> ist kein Palindrom". Dabei ist für <*eingabe*> das jeweils vom Benutzer eingegebene Wort zu einzusetzen.

Aufgabe 4.3 ***(5 Punkte)***
Die Sinusfunktion kann durch die folgende Potenzreihenentwicklung ausgedrückt werden:

$$\sin(x) = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n+1}}{(2n+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \ldots$$

Schreiben Sie ein rein funktionales, rekursives Programm, das vom Benutzer als Eingabe einen Wert x mit $0 \leq x \leq 2\pi$ einliest und als Ergebnis *sin(x)* und *cos(x)* ausgibt. Schreiben Sie eine Funktion für die Berechnung der Fakultät, eine Funktion für die Berechnung der Potenzen und eine Funktion für das Aufaddieren der Terme. Abbruchbedingung der Rekursion: Die Terme der Reihenentwicklung sollen aufaddiert werden bis der Betrag des nächsten zu addierenden Terms kleiner als 10^{-9} ist. Verwenden Sie für alle Berechnungen den Datentyp `LONGREAL`. Die Funktionen der mathematischen Bibliotheken von Modula-3 dürfen nicht verwendet werden.

Aufgabe 4.4 ***(5 Punkte)***
Schreiben Sie ein **imperatives** Programm, das eine Jahreszahl größer oder gleich 1582 einliest und berechnet, auf welchen Wochentag der Heiligabend in jenem Jahr fällt. Dazu sind folgende Informationen wichtig:

- Ein "normales" Jahr hat 365 Tage Tage.
- Ist die Jahreszahl durch 4 teilbar, aber nicht durch 100, hat es 366 Tage.
- Ist die Jahreszahl durch 100 teilbar, aber nicht durch 400, hat es wieder nur 365 Tage.
- Ist die Jahreszahl durch 400 teilbar, sind es doch 366 Tage.
- Der Heiligabend im Jahr 1582 war ein Freitag.

Hinweis: Die Einschränkungen, die hinsichtlich der erlaubten Befehle und Sprachkonstrukte bei den funktionalen Programmen gemacht wurden, gelten für diese Aufgabe nicht mehr! Sie dürfen also alle Befehle und Konstrukte verwenden, die Sie in der Vorlesung kennengelernt haben.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **5. Übungsblatt**

Ausgabe: Montag, 15.11.99 **Abgabe bis:** Montag, 22.11.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 25. und 26.11.99

Aufgabe 5.1 ***(10 Punkte)***
Gegeben sei folgender Ausschnitt aus einem Modula-3-Programm. Dieses Programm führt weder sinnvolle Berechnungen aus, noch ist es schön programmiert. Es soll nur zur Einübung von Gültigkeitsregeln dienen.

```
01 MODULE M EXPORTS Main;
02
03 VAR i, j, k : INTEGER;
04
05 PROCEDURE a (VAR i, k : INTEGER): INTEGER =
06
07     PROCEDURE b(VAR k : INTEGER; l : INTEGER): INTEGER =
08     BEGIN
09       i := i + 2 * l;
10       k := k - 1;
11       RETURN i;
12     END b;
13
14 BEGIN
15   j := k + 1 ;
16   i := k + b(i, j);
17   RETURN b(i, k);
18 END a;
19
20
21 PROCEDURE b (j : INTEGER; VAR k : INTEGER) =
22 BEGIN
23   j := j + k;
24   k := k + i;
25   i := i + j;
26 END b;
27
28 BEGIN
29   i := 3;
30   k := 2;
31   b( a(i, k), i);
32   i := a(i, k)
33 END M.
```

a) Versehen Sie die Bezeichner mit Indizes (z.B. `a1` statt `a`), so daß die Gültigkeitsbereiche der einzelnen Bezeichner deutlich werden. *(2 Punkte)*

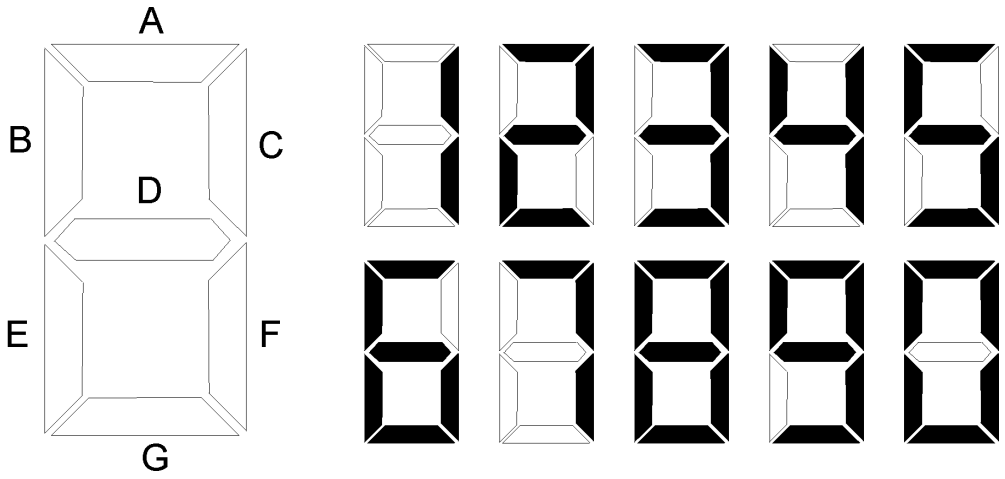
b) Als Sichtbarkeitsbereich eines Bezeichners bezeichnet man die Einheiten eines Programms, in denen dieser Bezeichner benutzt werden kann. Das ist also der Gültigkeitsbereich dieses Bezeichners ohne die Programmteile, in denen der Bezeichner durch einen gleichnamigen anderen Bezeichner verdeckt ist. Identifizieren Sie in dem oben angegebenen Programmstück, die Sichtbarkeitsbereich für die folgenden Bezeichner:

`i` in Zeile 3, `k` in Zeile 3, `k` in Zeile 5, `k` in Zeile 7, `j` in Zeile 21 *(2 Punkte)*

c) Geben Sie zu folgenden angewandten Bezeichnern aus dem obigen Programmstück jeweils das definierende Auftreten an:
`k` in Zeile 10, `j` in Zeile 15, `i` in Zeile 17, `j` in Zeile 23, `i` in Zeile 25 *(2 Punkte)*

d) Zeichnen Sie das Ablaufdiagramm mit der Lebensdauer der Variablen (gemäß dem Beispiel aus der Vorlesung) und ermitteln Sie damit alle Zwischen- und Endwerte der Variablen. *(4 Punkte)*

Aufgabe 5.2 ***(5 Punkte)***
Bei dem 7-Segment-Code für eine Taschenrechner-Sichtanzeige sollen die einzelnen Segmente für die Darstellung der Ziffern 0-9 angesteuert werden. Schreiben Sie ein Programm, das eine positive, ganze, maximal 5-stellige Zahl einliest und als Ausgabe eine Folge von Bezeichnern der im Display anzuschaltenden Segmente angibt. Hinweis: Wenn die eingegebene Zahl *n* Ziffern hat, sollen auch *n* Segmentfolgen zeilenweise ausgegeben werden, beginnend mit der Folge für die Ziffer, die den höchsten Stellenwert hat. Die Segmente werden wie folgt bezeichnet:



Aufgabe 5.3 ***(5 Punkte)***
Schreiben Sie ein Programm zum Rechnen mit Uhrzeiten. Das Programm soll als Eingabe zuerst einzeln die Stunden und Minuten der aktuellen Uhrzeit einlesen. Dann soll das Programm einzeln die Stunden und Minuten einer Zeitspanne einlesen, die zu der aktuellen Uhrzeit hinzuaddiert werden soll. Das Programm soll die Endzeit (eingegebene Uhrzeit plus eingegebene Zeitspanne) in Stunden und Minuten als Ergebnis ausgeben.
Beispielablauf des Programms für die Eingabe 16.50 Uhr und 1h 20min:

```
Bitte geben Sie die Stunden der Uhrzeit ein : 16
Bitte geben Sie die Minuten der Uhrzeit ein : 50
Wieviele Stunden weiter                     : 1
Wieviele Minuten weiter                     : 20

In 1 h und 20 min von 16.50 Uhr an gerechnet ist es 18.10 Uhr.
```

Lösen Sie die Aufgabe, indem Sie eine Prozedur schreiben, die als Eingabeparameter die vier vom Benutzer eingegebenen Werte erhält (Stunden der Uhrzeit, Minuten der Uhrzeit, Stunden weiter, Minuten weiter) und das Ergebnis, die berechnete Uhrzeit, in zwei Ausgabeparametern (einer für die Stunden, einer für die Minuten) zurückgibt.
Hinweis: Die Stunden und Minuten des Ergebnisses sind jeweils Referenzparameter der Prozedur.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung — WS 1999/2000 — 6. Übungsblatt

Ausgabe: Montag, 22.11.99 **Abgabe bis:** Montag, 29.11.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 2. und 3.12.99

Aufgabe 6.1 ***(5 Punkte)***
Formulieren Sie das Programm zu Aufgabe 3.4, das alle Vokale eines Textes in Unterstriche umsetzt, in imperativer Form (ohne Verwendung von Rekursion). Erläutern Sie die Unterschiede zur rekursiven Lösung. Welche Umformungsschritte wurden im einzelnen gemacht?

Aufgabe 6.2 ***(5 Punkte)***

a) Schreiben Sie Funktionsprozeduren, die jeweils für ganzzahliges a, b ($b \geq 0$), die Funktion a^b berechnen. Verwenden Sie nur die Operationen +, – und * und realisieren Sie die Iterationen durch folgende Schleifen: *(3 Punkte)*
   - eine `FOR`-Schleife
   - eine `WHILE`-Schleife
   - eine `REPEAT-UNTIL`-Schleife
   - eine `LOOP`-Schleife

b) Diskutieren Sie kurz die Vor- und Nachteile der einzelnen Lösungen. Welche Schleifenform ist für dieses Problem die angemessene? *(2 Punkte)*

Aufgabe 6.3 ***(5 Punkte)***
Schreiben Sie ein Programm, das den durchschnittlichen Kraftstoffverbrauch von bestimmten Fahrzeugen berechnet. Dem Programm sind folgende Fahrzeuge bekannt: Fahrrad, Skateboard, Mofa, Moped, Motorrad, Pkw, Lkw, Bus. Der Verbrauch pro 100 *km* bestimmt sich wie folgt:

- Nichtmotorisierte Fahrzeuge brauchen keinen Treibstoff, aber der Fahrer benötigt 0,1 *l* Milch pro angefangene 10 *kg* Körpergewicht. (Milch soll in diesem Falls als „Kraftstoff“ gelten.)
- Motorisierte Zweiräder benötigen 4 *l* Treibstoff plus 0.1 *l* Treibstoff je angefangene 10 *kg* Körpergewicht.
- Pkws benötigen 6 *l* Treibstoff plus ½ n *l*, wobei n die Zahl der Insassen ist.
- Lkws benötigen 15 *l* Treibstoff plus 1.5 *ln*(k) *l* , wobei k die Zuladung in Tonnen und *ln* der natürliche Logarithmus ist.
- Busse benötigen 17 *l* Treibstoff plus (1.3) m *l*, wobei m die Zahl der angefangenen Dutzend Insassen ist.

Das Programm soll die Fahrzeugart einlesen sowie die benötigten Berechnungswerte (Gewicht des Fahrers, Zahl der Insassen, Gewicht der Zuladung) und den benötigten Treibstoffverbrauch ausgeben. Verwenden Sie für die Fahrzeugarten einen Aufzählungsdatentyp.

Aufgabe 6.4 ***(5 Punkte)***

a) Entwerfen Sie einen Modula-3 Datentyp, der einen Fahrzeugschein für Kraftfahrzeuge beschreibt. Folgende Daten sollen in den Schein aufgenommen werden: *(3 Punkte)*

- das Kennzeichen – 1 bis 3 Buchstaben für den Ort, dann 1 bis 2 Buchstaben, dann 1 bis 4 Ziffern
- die Herstellerfirma
- die Fahrzeugnummer
- die Leistung in *kW*
- das Zulässige Gesamtgewicht
- falls es sich um ein Motorrad handelt: der Hersteller und die Zulassungsnummer des Schalldämpfers (eine positive ganze Zahl)
- falls es sich um einen Pkw oder Lkw handelt: die Maße (Länge, Höhe, Breite) und die Zahl der Achsen
- falls es sich um einen Lkw handelt: die Nutzlast und der Rauminhalt des Tanks in m^3

b) Schreiben Sie eine Prozedur, die diesen Datentyp durch Benutzereingaben mit sinnvollen Werten füllt. *(2 Punkte)*

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

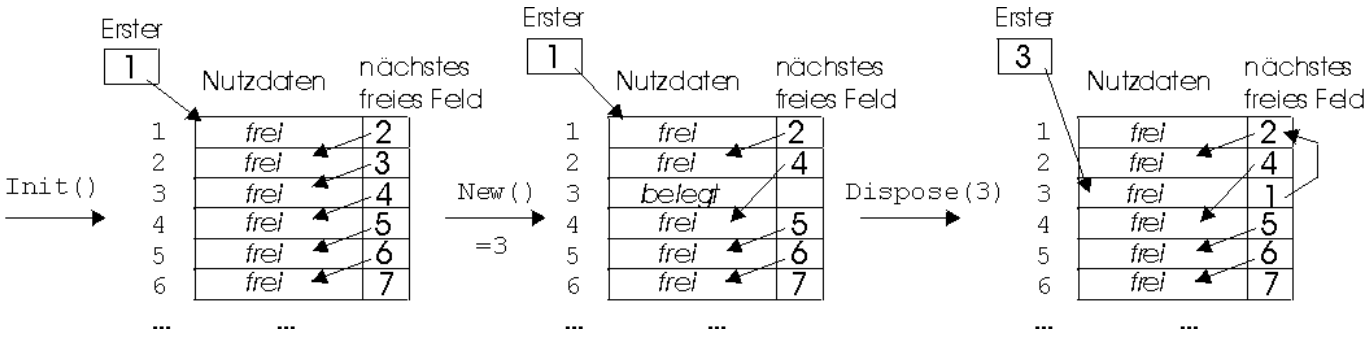
Programmierung **WS 1999/2000** **7. Übungsblatt**

Ausgabe: Montag, 29.11.99 **Abgabe bis:** Montag, 6.12.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 9. und 10.12.99

Aufgabe 7.1 ***(10 Punkte)***

Es gibt Programmiersprachen, die im Gegensatz zu Modula-3, keine Zeiger kennen. Wenn man in diesen Sprachen dynamische Datenstrukturen implementieren will, muß man die Verwaltung des freien Speichers, der Halde, selber implementieren. Die Halde wird als ein Array implementiert. Die Elemente des Arrays bestehen aus zwei Komponenten, von denen eine das Nutzdatum enthält und die andere den Index des nächsten freien Elements (Achtung: Das ist kein Zeiger, sondern ein Integerwert!). Am Anfang wird die Halde initialisiert, indem in jedem Haldenelement als Index des nächsten freien Elements der Index des nächsten Elements im Array eingetragen wird. Zur Implementierung der Halde gehört zusätzlich noch eine Variable, die den Index des ersten freien Elements der Halde enthält.



Das Anfordern von Speicherplätzen aus Halde geschieht über eine Prozedur `New`, die als Ergebnis den Index eines freien Elements zurückgibt. Dieses Element wird dabei aus der Liste der freien Elemente der Halde herausgenommen und kann im Programm verwendet werden. (Anmerkung: Um die Verkettung der Haldenelemente anschaulich zu machen, wurde in der Darstellung oben das Element 3 aus der Mitte der Halde beim Aufruf von `New` zurückgeliefert. Im Normalfall nimmt man einfach das erste freie Element!) Sobald ein Element der Halde nicht mehr gebraucht wird, wird es durch Aufruf einer Prozedur `Dispose` wieder in die Liste der freien Haldenelemente eingefügt.

a) Definieren Sie eine Datenstruktur für Haldenelemente, bei denen Integerwerte als Nutzdaten verwaltet werden. Realisieren Sie eine Halde mit 10 Elementen wie oben beschrieben als Array und schreiben Sie die Prozeduren `Init`, `New` und `Dispose` für das Initialisieren der Halde, sowie das Belegen und Freigeben von Speicher. *(5 Punkte)*

b) Schreiben Sie ein Prozedur `PrintHeap`, die den Inhalt der Halde in der Reihenfolge der Speicherplätze ausgibt. Kennzeichnen Sie dabei die freien Felder der Halde! *(2 Punkte)*

c) Schreiben Sie ein Hauptprogramm, in dem Sie nacheinander 4 Haldenelemente durch Aufruf von `New` belegen, daraus eine Liste aus 4 Integerwerten aufbauen und anschließend mit `Dispose` wieder freigeben. Geben Sie die Elemente in der selben Reihenfolge frei, in der Sie sie belegt haben. Geben Sie nach jedem Aufruf von `New` und `Dispose` die Halde mit der Prozedur `PrintHeap` aus. Geben Sie zusammen mit dem Programmlisting auch ein Protokoll der Ausgaben des Programms ab. *(3 Punkte)*

Aufgabe 7.2 ***(10 Punkte)***

a) Entwerfen Sie ein Datenstruktur, mit der Sie eine sortierte, lineare, doppelt verkettete Liste aufbauen können, in der Adressen gespeichert werden. Verwenden Sie Zeiger! Eine Adresse besteht aus Name, Straße, Hausnummer, Postleitzahl und Ort. Hinweis: Definieren Sie einen `RECORD`-Typ für die Speicherung der Adresse. *(2 Punkte)*

b) Schreiben Sie jeweils eine Prozedur zum Einfügen eines neuen Elements in die Liste, zum Suchen nach einem Element in der Liste und zum Löschen eines Elements aus der Liste. Die Liste soll nach den Namen in alphabetisch aufsteigender Reihenfolge sortiert sein. Falls ein Name in die Liste eingefügt werden soll, der bereits in der Liste vorhanden ist, wird der vorhandene Eintrag durch den neuen Eintrag ersetzt. Die Prozedur zum Suchen in der Liste erhält als einen Eingabeparameter einen Namen (von Typ `TEXT`) und sucht in der Liste nach dem Element mit diesem Namen. Wenn in der Liste der Name gefunden wird, soll die Prozedur einen Zeiger auf dieses Element der Liste zurückgeben. Wenn der Name nicht in der Liste gefunden wird, soll als Ergebnis `NIL` zurückgegeben werden. *(5 Punkte)*

c) Schreiben Sie ein Hauptprogramm zu diesen beiden Funktionen. In dem Programm soll dem Benutzer in einem Menü die Wahl zwischen folgenden Aktionen haben: *(3 Punkte)*

1. Eingabe einer neuen Adresse: Die eingegebene Adresse wird anschließend in die Liste eingefügt.
2. Anzeigen der gesamten Liste: Die Einträge werden untereinander ausgegeben.
3. Suchen nach einem Namen in der Liste: Wenn der Name in der Liste vorhanden ist, wird der entsprechende Eintrag ausgegeben.
4. Löschen eines Elements aus der Liste: Wenn der Name in der Liste vorhanden ist, wird das entsprechende Element aus der Liste entfernt.
5. Beenden des Programms: Das Programm wird nur dann beendet, wenn der Benutzer diese Aktion auswählt. Nach allen anderen Aktion kehrt das Programm wieder ins Menü zurück.

Aufgabe 7.3 Zusatzaufgabe ***(5 Bonuspunkte)***

Der Weihnachtsmann steht jedes Jahr vor dem selben Problem: Durch die weltweite Bevölkerungsexplosion kann er mit seinem Schlitten einfach nicht mehr die Geschenke für alle Menschen auf einer Tour ausliefern, sie sind zu schwer. Er muß daher mehrmals fahren und zwischendurch immer wieder zum Nordpol zurückkehren und dort wieder neue Pakete aufladen. Leider ist der Nordpol auch nicht mehr das was er mal war und im vergangenen Jahr wurden einige wertvolle Pakete, die der Weihnachtsmann am Nordpol zurückgelassen hatte, in seiner Abwesenheit geklaut. Daher hat der Weihnachtsmann beschlossen, in diesem Jahr die wertvollen Pakete zuerst auszuliefern und den Gesamtwert der zurückgelassenen Pakete möglichst gering zu halten.
Helfen Sie dem Weihnachtsmann, indem Sie ein **möglichst effizientes** Programm schreiben, daß für eine gegebene Menge an Paketen, die jeweils einen Wert und ein Gewicht haben, bestimmt, welche Kombination dieser Pakete den größten Wert hat und die maximale Zuladung des Schlittens nicht überschreitet. Der Schlitten des Weihnachtsmanns hat, laut Fahrzeugpapieren, eine maximale Zuladung von 1000 kg. Testen Sie das Programm für die folgende Menge von Paketen. Die Werte in den Klammern bezeichnen den Wert und das Gewicht (*DM*, *kg*) der einzelnen Pakete:

(100,299)	(120,150)	(200, 170)	(80, 180)	(320, 300)	(65, 254)	(120, 275)
(55,175)	(20,140)	(65, 190)	(130, 180)	(120, 240)	(120, 250)	(200, 500)
(89,200)	(150, 260)	(230, 405)	(120, 190)	(110, 210)	(202, 405)	(112, 266)
(150,190)	(290,220)	(185, 420)	(140, 310)	(90, 180)	(60, 210)	(202, 202)

Hinweis: Sie brauchen die Daten der Pakete nicht zur Laufzeit eingeben. Verwenden Sie statt dessen ein statisch initialisiertes Array.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung WS 1999/2000 8. Übungsblatt

Ausgabe: Montag, 6.12.99 **Abgabe bis:** Montag, 13.12.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 16. und 17.12.99

Aufgabe 8.1 ***(5 Punkte)***

In der Vorlesung wurden drei Varianten von *Bubblesort* vorgestellt. Realisieren Sie jede dieser Varianten in einer eigenen Prozedur. Realisieren Sie die Prozeduren so, daß die Anzahl der vorgenommenen Vertauschungen gezählt und ausgegeben wird. Schreiben Sie ein Hauptprogramm, welches ein Feld der Länge 1000 enthält. Initialisieren Sie dieses Feld zu einem mit einer bereits sortierten Folge und zum anderen mit Hilfe eines Zufallszahlengenerators. Lassen Sie dieses Feld in beiden Fällen mit jeder der drei Prozeduren sortieren. Stellen Sie die Anzahl Vertauschungen, die jede Prozedur durchführt, gegenüber und diskutieren Sie das Ergebnis. (Hinweis: Ein Zufallszahlengenerator wird durch das Modul `Random` zur Verfügung gestellt. Auf Webseite der Vorlesung befindet sich ein Beispielprogramm dazu.)

Aufgabe 8.2 ***(7 Punkte)***

a) Schreiben Sie eine Prozedur *Mergesort*, die eine einfach verkettete Liste von `CARDINAL`-Zahlen in der folgenden Weise sortiert: Zunächst wird die Eingabefolge in zwei beliebige Teilfolgen geteilt. Jede dieser beiden Teilfolgen wird, wenn sie aus mehr als einem Element besteht, nun für sich alleine durch *Mergesort* sortiert. *Mergesort* arbeitet also rekursiv. Zum Schluß mischt man die sortierten Teilfolgen durch Vergleich der Elemente der Teilfolgen zur sortierten Ergebnisfolge zusammen. Beim Mischen vergleicht man zunächst die beiden größten Elemente jeder Teilfolge. Das größte dieser beiden Elemente wird dann das größte Element der Ergebnisfolge. Anschließend vergleicht man die beiden größten Elemente aus beiden Teilfolgen, die noch nicht in die Ergebnisfolge übertragen wurden, und nimmt davon wiederum das größte Element usw.
Schreiben Sie zum Testen ein Programm, daß die folgende Zahlenfolge einliest, mit der Prozedur *Mergesort* sortiert und schließlich das Ergebnis wieder ausgibt: *(5 Punkte)*

15, 22, 7, 14, 48, 25, 20, 29, 9, 18, 38, 51, 27, 11, 31

b) Skizzieren Sie die Vorgänge beim Sortieren durch Mischen (*Mergesort*) anhand der oben angegebenen Zahlenfolge, indem Sie die internen Zwischenzustände der Zahlenfolge in geeigneter Form darstellen *(2 Punkte)*

Aufgabe 8.3 ***(8 Punkte)***

Die folgende Modula-3-Prozedur `Div` berechnet die ganzzahlige Division zweier natürlicher Zahlen unter ausschließlicher Verwendung von Addition und Subtraktion. (Sie berechnet also das gleiche wie die Standard-Operation `DIV`)

```
PROCEDURE Div (x, y: CARDINAL; VAR z: CARDINAL) =
VAR r: CARDINAL;
BEGIN
  z := 0;
  r := x;
  WHILE r >= y DO
    r := r-y;
    z := z+1;
  END;
END Div;
```

Zeigen Sie die partielle Korrektheit von `Div`. Genauer gesagt, zeigen Sie, daß

$$\{(x \geq 0) \wedge (y > 0\} \text{ Div } \{(x = z*y+r) \wedge (0 \leq r < y)\}.$$

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung — WS 1999/2000 — 9. Übungsblatt

Ausgabe: Montag, 13.12.99 **Abgabe bis:** Montag, 20.12.99, 13.30 Uhr

Besprechung: in den Übungsgruppen am 23.12.99 und 7.1.00

Aufgabe 9.1 ***(6 Punkte)***
Der folgende Algorithmus beschreibt ein weiteres Sortierverfahren, das „Sortieren durch Einfügen". Er sortiert das Feld `a: ARRAY [1..n] OF CARDINAL` in aufsteigender Reihenfolge.

```
FOR i:=2 TO n DO
  x := a[i];
  L := 1;
  R := i;

  WHILE L<R DO
    m := (L+R) DIV 2
    IF a[m] <= x THEN
      L := m+1;
    ELSE
      R := m;
    END;
  END;

  (* Einfuegestelle R gefunden *)
  FOR j := i TO R+1 BY -1 DO
    a[j] := a[j-1];
  END;
  a[R] := x;
END;
```

Führen Sie einen Whitebox-Test für das obige Programmstück durch. Stellen Sie den Flußgraphen für diesen Algorithmus auf und geben Sie geeignete Testfälle an, die (zusammengenommen) alle Pfade des Flußgraphen durchlaufen.

Aufgabe 9.2 ***(6 Punkte)***
Zur Beschreibung des Aufwands von Algorithmen gibt es die sogenannte O-Notation. Die Definition lautet wie folgt:

Sei $f : N_0 \to N_0$, dann gilt:

$$O(f) := \{g : N_0 \to N_0 \mid \exists c > 0, \exists n_0 \in N_0 : g(n) \leq cf(n) \forall n \geq n_0\}$$

Statt z.B. $O(f : N_0 \to N_0$ mit $f(n) = n^2)$ wird meist die abkürzende Schreibweise $O(n^2)$ verwendet.
In der Vorlesung wurde gezeigt, daß der Bubblesort-Algorithmus in $O(n^2)$ ist, also einen Aufwand hat, der quadratisch mit der Länge der zu sortierenden Folge wächst. Betrachtet werden bei Effizienzbetrachtungen von Sortieralgorithmen meist die Zahl der Vergleiche und die Zahl der Elementbewegungen (wobei in der Regel davon abstrahiert wird, ob tatsächlich das Datenelement oder nur ein Zeiger darauf bewegt wird).

a) Schätzen Sie den Aufwand für den Algorithmus „Sortieren durch Einfügen" aus Aufgabe 9.1 ab. *(3 Punkte)*

b) Schätzen Sie den Aufwand für das „Sortieren durch Mischen" (Mergesort) aus Aufgabe 8.2 ab. *(3 Punkte)*

Aufgabe 9.3 ***(8 Punkte)***
Sie wollen ein Programm zur Beratung von Kapitalanlegern schreiben. Folgende Informationen stehen Ihnen aus der Finanzwelt zur Verfügung:

1. Die Anlage auf einem Sparbuch mit 3-monatiger Kündigungsfrist bringt jährlich 3,5% Zinsen. Das Risiko dieser Anlageform ist äußerst gering. Es können Beträge in beliebiger Höhe angelegt werden.
2. Die Anlage in festverzinslichen Wertpapieren mit 1-jähriger Laufzeit bringt jährlich 8,0% Zinsen. Das Risiko ist gering. Es müssen mindestens DM 5000 angelegt werden.
3. Die Anlage in festverzinslichen Wertpapieren mit 3-jähriger Laufzeit bringt jährlich 6,5% Zinsen. Das Risiko ist gering. Es müssen mindestens DM 1000 angelegt werden.
4. Die Anlage in Aktien der Firma „*Dream Company & Co*“ (Stückpreis DM 200) bringt einen geschätzten Wertzuwachs pro Jahr um 9,0%. Das Risiko dieser Anlage ist sehr hoch. Es müssen mindestens fünf Aktien gekauft werden, die allerdings jederzeit wieder verkauft werden können.
5. Die Anlage in Aktien der Firma „*Security Company & Co*“ (Stückpreis DM 500) bringt einen geschätzten Wertzuwachs pro Jahr um 6,0%. Das Risiko dieser Anlage liegt im mittleren Bereich. Die Aktien können jederzeit wieder verkauft werden.
6. Die Anlage in einem Investmentfond bringt einen geschätzten jährlichen Wertzuwachs von 8,0%. Das Risiko dieser Anlage ist im mittleren Bereich, da das Kapital auf verschiedene Firmen verteilt wird. Es müssen mindestens DM 5000 angelegt werden.

a) Entwickeln Sie eine Entscheidungstabelle, die die verschiedenen Wünsche der Anleger berücksichtigt:

   - die geforderte Höhe des Ertrags (= jährliche Verzinsung) in drei Stufen ($x < 4\%$; $4\% \leq x < 7\%$; $x \geq 7\%$) in Verbindung mit der Risikobereitschaft des Anlegers (hoch – mittel – niedrig),
   - die Verfügbarkeit des angelegten Kapitals (innerhalb eines Jahres oder erst nach Ablauf eines Jahres),
   - die Höhe des Startkapitals ($x < 1000$; $1000 \leq x < 5000$; $x \geq 5000$) .

   Die Entscheidungstabelle gibt dann in Abhängigkeit von den Kundenwünschen an, welche Anlageformen in Betracht kommen. *(4 Punkte)*

b) Schreiben Sie ein Modula-3 Programm, das die Kunden gemäß der Entscheidungstabelle berät. Dieses Programm erhält als Eingabedaten also die Informationen über die vom Kapitalanleger geforderten Bedingungen und liefert als Ergebnis Informationen, welche der sechs Anlagemöglichkeiten in diesem Fall in Frage kommen. *(4 Punkte)*

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **10. Übungsblatt**

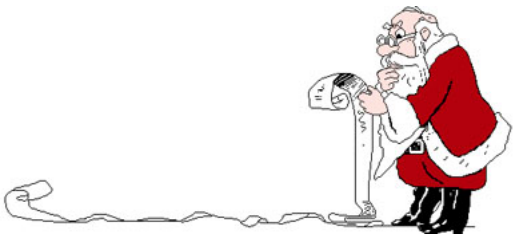
Ausgabe: Montag, 20.12.99 **Abgabe bis:** Montag, 10.1.00, 13.30 Uhr

Besprechung: in den Übungsgruppen am 13. und 14.1.00

Aufgabe 10.1 ***(7 Punkte)***
Für die wohlbekannte Datenstruktur Keller soll die Schnittstelle eines Datenobjektmodul definiert und der Rumpf des Moduls implementiert werden. Bei der Schnittstellengestaltung führen Sie Sicherheitsabfragen ein.
Die Realisierung ist in einem Feld (Cursorrealisierung) vorzusehen. Dieses Feld enthält neben abzulegenden Kellerelementen auch die Freispeicherliste. Achten Sie im Rumpf auf eine saubere Definition der Daten mittels Typdeklarationen. Alle Zugriffsoperationen des ADO-Moduls sind zu implementieren. Insbesondere ist eine Initialisierungsoperation in der Schnittstelle vorzusehen und im Rumpf zu realisieren.

Aufgabe 10.2 ***(7 Punkte)***
Implementieren Sie den in der Vorlesung vorgestellten abstrakten Datentyp Stack (Kellerspeicher), der auf der Folie 19 im Teil IV - Datenabstraktion spezifiziert ist. Realisieren Sie den Stack mit Hilfe einer linearen Liste. Schreiben Sie ein Modul Stack, das den Datentyp und die auf Folie 19 angegebenen Operationen auf Stacks an seiner Schnittstelle zur Verfügung stellt. An der Schnittstelle darf nicht erkennbar sein, daß der Stack intern mit einer linearen Liste implementiert ist. Schreiben Sie ein weiteres Modul, das den Stack verwendet. In diesem Modul soll dem Benutzer ein Menü angeboten werden, mit dem er einen neuen Stack anlegen, eine Datum auf den Stack schreiben, ein Datum vom Stack herunternehmen, das oberste Element des Stacks ansehen und das Programm verlassen kann. Geben Sie zusammen mit den Quelltexten der Module auch das m3makefile ab!

Aufgabe 10.3 ***(6 Punkte + 3 Extrapunkte für Zusatz)***



Da der Weihnachtsmann wie jedes Jahr im Dezember sehr ausgelastet ist, bittet er Sie, ein Programm zu entwickeln, das ihm seine Arbeit an den Weihnachtsfeiertagen erheblich erleichtern soll.
Der Weihnachtsmann muß in vielen Städten Deutschlands Geschenke überreichen und überlegt, wie er seine Reise wohl am sinnvollsten plant. Um möglichst schnell alle Städte erreichen zu können, soll Ihr Programm für ihn die kürzeste Reiseroute berechnen, auf der er alle Städte einmal erreicht. Der Start der Route soll in einer der Städte liegen und am Ende der Route soll der Weihnachtsmann wieder zum Ausgangspunkt zurückkehren. Als Hilfsmittel sind dem Programm die Entfernungen zwischen den Städten bekannt: Aachen (AC), Bamberg (BA), Berlin (B), Bonn (BN), Düsseldorf (D), Essen (E), Hamburg (HH), Köln (K), München (M), Stuttgart (S).

	AC	BA	B	BN	D	E	HH	K	M
BA	455	---							
B	644	404	---						
BN	94	378	633	---					
D	79	417	571	70	---				
E	117	450	527	103	38	---			
HH	486	517	285	454	407	369	---		
K	72	374	570	27	43	76	427	---	
M	628	240	592	558	628	643	757	567	---
S	480	244	624	362	403	436	690	360	218

Hinweis: Die vom Weihnachtsmann zurückgelegte Route soll einen Kreis darstellen, so daß jede Stadt genau einmal besucht wird, bevor er zum Ausgangspunkt zurückkehrt. Dieses Problem ist als *Traveling-Salesman-Problem* bekannt. Es fällt in dieselbe Klasse von Problemen wie das der Zusatzaufgabe auf Übungsblatt 7, d.h. es gibt ebenfalls keinen effizienten Algorithmus zu dessen optimaler Lösung.

Zusatzpunkte gibt es, wenn die Datenstruktur für die Entfernung der Städte und die Verwaltungsinformation der Routenfindung als ADO-Baustein abgelegt wird.

Frohe Weihnachten und ein gutes neues Jahr

bitte Rückseite beachten

Zusatzaufgabe (7 Punkte) *Durch diese Aufgabe ändert sich die 50%-Grenze nicht!*

Schreiben Sie ein selbstlernendes Tierrateprogramm, mit dem etwa der folgende Dialog möglich ist:
1. Spiel (Noch kennt das Programm nur den Elefanten.):

```
Programm:   Ist es ein Elefant?
Benutzer:   Nein
P:          Du hast gewonnen. Ich weiß nicht weiter. Welches Tier war es?
B:          Regenwurm.
P:          Oh, den kannte ich nicht! Sag doch bitte eine Frage, durch die
            ich die beiden unterscheiden kann.
B:          Ist es ein Wirbeltier?
P:          Wird die Frage für mein geratenes Tier mit JA beantwortet?
B:          Ja
P:          Danke! Das nächste Mal bin ich schlauer!
```

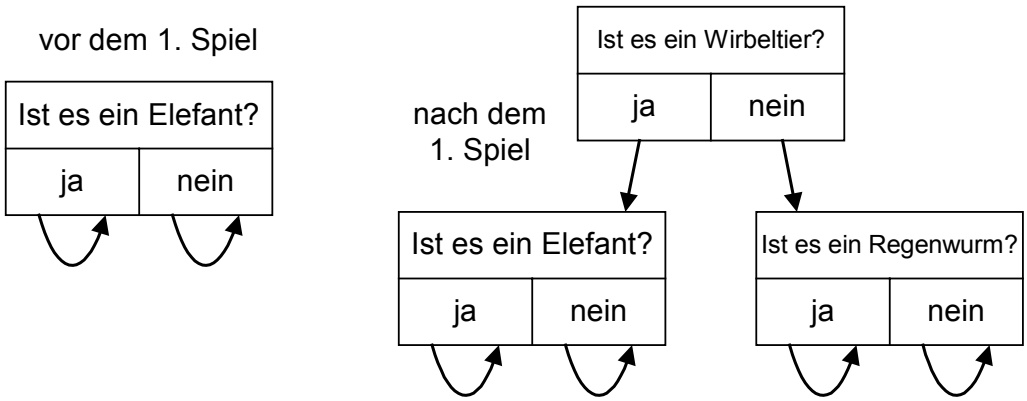
2. Spiel (Nun ist das Programm schon etwas schlauer.):

```
P:          Ist es ein Wirbeltier?
B:          Ja
P:          Ist es ein Elefant?
B:          Nein
P:          Du hast gewonnen. Ich weiß nicht weiter. Welches Tier war es?
B:          Spatz
P:          Oh, den kannte ich nicht! Sag doch bitte eine Frage, durch die
            ich die beiden unterscheiden kann.
B:          Ist es ein Säugetier?
P:          Wird die Frage für mein geratenes Tier mit JA beantwortet?
B:          Ja
P:          Danke! Das nächste Mal bin ich schlauer!
```

3. Spiel (Das Programm wird immer schlauer.):

```
P:          Ist es ein Wirbeltier?
B:          Ja
P:          Ist es ein Säugetier?
B:          Nein
P:          Ist es ein Spatz?
B:          Nein
P:          Du hast gewonnen. Ich weiß nicht weiter. Welches Tier war es?
B:          Adler
P:          Oh, den kannte ...
```

Zuerst muß man ein Datenobjektmodul Ratebaum erstellen. Ein Element dieses Baumes besteht aus einer Frage und zwei Unterbäumen, einem für die Antwort JA und einem für die Antwort NEIN. Man muß in diesem Baum navigieren, Fragen an Blättern ersetzen und neue Bäume an Blätter anhängen können.



Danach muß ein ausführbares Modul geschrieben werden, daß unter Benutzung dieses Ratebaums den Dialog mit dem Benutzer führt. Geben Sie zusätzlich zu den Quelltexten der Module das m3makefile ab!

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung — WS 1999/2000 — 11. Übungsblatt

Ausgabe: Montag, 10.1.00 **Abgabe bis:** Montag, 17.1.00, 13.30 Uhr

Besprechung: in den Übungsgruppen am 20. und 21.1.00

Aufgabe 11.1 ***(10 Punkte)***
In Aufgabe 7.2 haben Sie ein Programm zur Adressenverwaltung geschrieben. Die Adressen werden dabei in einer doppelt verketteten Liste gespeichert. Der Benutzer kann über ein Menü die verschiedenen Operationen auf dem Datenbestand aufrufen, wie z.B. Einfügen, Suchen, etc. Die Definition der Liste und die Zugriffsoperationen standen dabei im selben Modul wie das Hauptprogramm und die Hilfsprozeduren, die die Liste verwenden. Da Sie inzwischen ADTs und die Aufteilung eines Programms auf mehrere Module kennengelernt haben, sollen Sie das Programm aus Aufgabe 7.2 neu strukturieren und verbessern. Das Programm soll auf drei Module aufgeteilt werden. Gehen Sie dazu von der Musterlösung der Aufgabe aus. Geben Sie auch das m3makefile ab.

a) Schreiben sie ein Modul `AddressStorage`, das einen abstrakten Datentyp `AdrStorage` zur Verwaltung von Adressen implementiert. Intern sollen die Adressen wie bisher in einer doppelt verketteten Liste gespeichert werden. Das Modul soll die Typen `Address` und `AdrStorage` exportieren. Realisieren Sie `AdrStorage` als einen opaken Referenztyp! Darüber hinaus sollen die folgenden Prozeduren exportiert werden: *(5 Punkte)*

```
Init      (VAR adrst:AdrStorage);
Insert    (VAR adrst:AdrStorage; newAdr:Address);
Find      (    adrst:AdrStorage; name:Text): Address;
Delete    (VAR adrst:AdrStorage; name:Text);
PrintAll  (    adrst:AdrStorage);
```

b) Schreiben Sie ein Modul `AddressOperations`, das die Hilfsfunktionen `InputAddress`, `PrintList`, `SearchAddress` und `RemoveName` exportiert und ein Hauptmodul, das das Auswahlmenü anbietet, über das der Benutzer diese Funktionen aufruft. *(5 Punkte)*

Aufgabe 11.2 ***(10 Punkte)***
Im Interface eines Moduls steht für jede exportierte Prozedur eine Deklaration. Diese Deklaration entspricht dem Prozedurkopf und wird mit einem Semikolon beendet. Um für ein Modul ein Interface zu erstellen, ist es am einfachsten, wenn man im Texteditor die Prozedurköpfe aus der Implementierung (.m3-Datei) kopiert und nacheinander in das Interface (.i3-Datei) einfügt.
Schreiben Sie ein Programm, mit dem Sie das Erstellen eines Interfaces für eine gegebene Implementierung unterstützen. Das Programm soll als erstes den Benutzer nach dem Namen des Moduls fragen. Danach soll es die Datei <*name*>`.m3` öffnen und Wort für Wort lesen. Sobald in der Datei das Schlüsselwort `PROCEDURE` gefunden wird, soll das Programm den kompletten Kopf der Prozedur lesen und den Benutzer fragen, ob er diese Prozedur im Interface exportieren möchte. Aus den Prozeduren, für die der Benutzer diese Frage mit ja beantwortet hat, soll das Programm eine syntaktisch korrekte Interface-Datei erzeugen, die ohne weitere Änderungen übersetzt werden kann. Hinweis: Taucht das Schlüsselwort `PROCEDURE` innerhalb eines Kommentars auf, soll es nicht beachtet werden. Ebenso sollen lokale Prozeduren ignoriert werden.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung WS 1999/2000 12. Übungsblatt

Ausgabe: Montag, 17.1.00 **Abgabe bis:** Montag, 24.1.00, 13.30 Uhr

Besprechung: in den Übungsgruppen am 27. und 28.1.00

Aufgabe 12.1 ***(8 Punkte)***
In einem objektorientierten Grafiksystem sollen die folgenden Klassen von grafischen Objekten vorhanden sein: Strecke, Streckenzug, Polygon, Kreis, Ellipse, Dreieck, Rechteck, Quadrat, Viereck.

a) Ordnen Sie diese Klassen in eine Klassenhierarchie ein und zeichnen Sie diese Hierarchie als einen Baum auf. Berücksichtigen Sie dabei, welche Klassen Spezialisierungen anderer Klassen sind und fügen Sie sinnvolle, gemeinsame Oberklassen in die Hierarchie ein. Schreiben Sie zu jeder Klasse hinzu, ob es sich um eine abstrakte oder konkrete Klasse handelt. *(4 Punkte)*

b) Das Grafiksystem soll die folgenden Operationen auf seinen Objekten anbieten:

Name	**Bedeutung**
`moveTo`	Verschiebe das Objekt an eine neue Position.
`position`	Ermittele die Position eines Objekts.
`length`	Ermittele die Länge des Objekts.
`area`	Ermittele die Fläche des Objekts.
`radius`	Ermittele den Radius des Objekts.
`lineColor`	Weise dem Objekt eine Linienfarbe zu.
`fillColor`	Weise dem Objekt eine Füllfarbe zu.
`boundingBox`	Ermittele das kleinste Rechteck, das das Objekt vollständig umschließt.
`perimeter`	Ermittele den Umfang des Objekts.

Ordnen Sie die oben aufgelisteten Operationen den einzelnen Klassen zu. Beachten Sie dabei, daß nicht jede Operation für jede Klasse sinnvoll ist! Schreiben Sie zu jeder Operation einer Klasse, ob sie konkret oder abstrakt ist. Konkrete Operationen sollen soweit oben wie möglich in der Klassenhierarchie definiert werden. *(4 Punkte)*

Aufgabe 12.2 ***(4 Punkte)***
In der Vorlesung haben Sie Zusicherungen (Assertions) und die Anweisungen `TRY` und `EXCEPT` kennengelernt, mit denen man Ausnahmen im Programmablauf behandeln kann. Verwenden Sie diese Anweisungen, um das Programm zum Addieren von Zeitspannen zur aktuellen Uhrzeit aus Aufgabe 5.3 gegen falsche Eingaben des Benutzers abzusichern. Gehen Sie dazu von der Musterlösung zu dieser Aufgabe aus. Stellen Sie mit Hilfe von Assertions und einer geeigneten Ausnahmebehandlung das folgenden Verhalten des Programms sicher:

- Wenn der Benutzer etwas anderes als eine Integerzahl eingibt, dann soll das Programm einen entsprechenden Hinweis ausgeben und den Benutzer erneut zur Eingabe auffordern.
- Das Programm soll nur sinnvolle Uhrzeiten akzeptieren, z.B. 21.55 Uhr. Bei nicht sinnvollen Uhrzeiten, z.B. 45.89 Uhr, soll der Benutzer ebenfalls aufgefordert werden, eine neue Uhrzeit einzugeben. (Hinweis: Die entsprechende Zusicherung soll in der Prozedur `AddTime` stehen).

Aufgabe 12.3 ***(8 Punkte)***

Der abstrakte Datentyp `AddressStorage`, den Sie in Aufgabe 11.1 implementiert haben, hat bis jetzt noch den Nachteil, daß der Typ eines Adreßeintrags im Interface als Record deklariert wird und ein Anwender des ADT direkt auf die einzelnen Komponenten einer Adresse zugreifen kann. Das Prinzip der Datenabstraktion wird damit verletzt. Lösen Sie dieses Problem indem Sie in einem Modul `Address` einen weiteren ADT zur Speicherung von Adressen implementieren und diesen in dem Modul `AddressStorage` verwenden.

Der ADT soll Funktionen zum Anlegen einer neuen Adresse und zum Lesen und Verändern der einzelnen Komponenten einer bestehenden Adresse anbieten. Um eine geordnete Liste von Adressen aufbauen zu können, benötigen Sie darüber hinaus noch eine Vergleichsoperation, mit der entschieden werden kann, ob eine Adresse alphabetisch größer ist als eine andere.

Ändern Sie das Modul `AddressOperations` sowie das Modul `AddressStorage` und dessen Prozeduren, so daß dort das Modul `Address` importiert und der ADT anstatt des bisher benutzten Records verwendet wird. Geben Sie sämtliche Module des Programms (Hauptmodul, Modul `Address`, Modul `AddressStorage`, Modul `AddressOperations`) und das m3makefile ab.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja|jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung **WS 1999/2000** **13. Übungsblatt**

Ausgabe: Montag, 24.1.00 **Abgabe bis:** Montag, 31.1.00, 13.30 Uhr

Besprechung: in den Übungsgruppen am 3. und 4.2.00

Aufgabe 13.1 ***(8 Punkte)***
In Aufgabe 12.1 wurde eine Klassenhierarchie für grafische Objekte erstellt. Dabei wurden den Klassen eine Reihe von abstrakten und konkreten Methoden zugeordnet. Implementieren Sie dieses Grafiksystem mit den Methoden `length`, `area`, `radius`, `boundingBox` und `perimeter`. Verwenden Sie dabei folgende interne Datenstrukturen für die einzelnen Klassen von graphischen Objekten:

- Ein Punkt wird durch eine Koordinate x und eine Koordinate y festgelegt
- eine Strecke durch zwei Punkte
- eine Streckenzug mit n Ecken durch ein Feld mit n+1 Elementen
- ein Polygon mit n Ecken durch ein Feld mit n Elementen
- ein Kreis durch seinen Mittelpunkt und seinen Radius
- eine Ellipse durch zwei Brennpunkte und den maximalen Durchmesser
- ein Dreieck durch drei Punkte
- ein Rechteck und ein Quadrat durch zwei Punkte
- ein Viereck durch vier Punkte

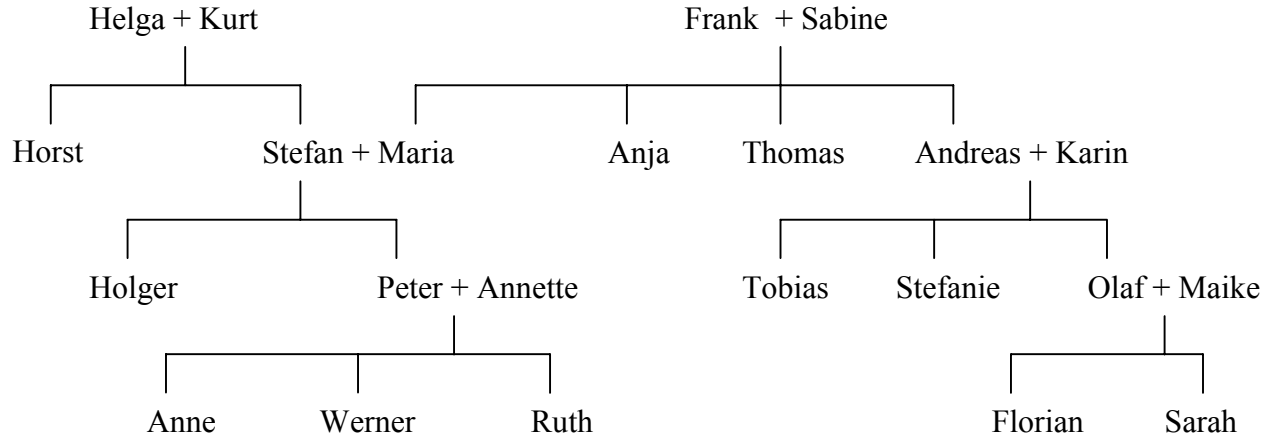
Aufgabe 13.2 ***(6 Punkte)***
Der abstrakte Datentyp `AddressStorage` ist eine Kollektion von Daten vom Typ `Address`. Der Datentyp `AddressStorage` soll derart verallgemeinert werden, so daß er nicht nur Adressen, sondern verschiedene Objekte speichern kann.

a) Geben Sie das Interface für einen allgemeinen Kollektionsspeicher `ObjectStorage` an, der Daten von dem allgemeinen Typ `StorableObject` aufnehmen kann. Geben Sie auch das Interface von `StorableObject` an. *(3 Punkte)*

In der Vorlesung „Beispiel einer Programmentwicklung“ wurde auf Folie 31 eine Klassenhierarchie für unterschiedliche Adreßarten vorgestellt.

b) Geben Sie das Interface für die Klassen von Folie 31 an. Verwenden sie dabei die Klassen `ObjectStorage` und `StorableObject`, so daß Objekte der Klasse `Adressen` (Wurzel der Klassenhierarchie auf Folie 31) in `ObjectStorage` abgelegt werden können. Geben Sie die Klassenhierarchie an, die die Klassen `ObjectStorage`, `StorableElement`, `Adressen`, `Geschäftsadressen` und `Privatadressen` umfaßt. *(3 Punkte)*

Aufgabe 13.3 ***(6 Punkte)***
Gegeben sei der folgende Stammbaum:



Schreiben Sie ein Programm in Prolog, mit dessen Hilfe Sie Verwandschaftsbeziehungen zwischen Personen in diesem Stammbaum bestimmen können. Definieren Sie in diesem Programm zuerst eine Menge von Basisfakten, die den Stammbaum wiedergeben. Führen Sie anschließend Regeln ein, um diejenigen Verwandschaftsbeziehungen zu ermitteln, die nicht bereits Teil der Basisfakten sind. Jede Regel soll dabei die Form *Beziehung(X,Y)* haben. Zum Beispiel soll *Tocher(X,Y)* genau dann wahr sein, wenn X eine Tochter von Y ist. Folgende Verwandschaftsbeziehungen sollen dem Programm bekannt sein:
Vater, Mutter, Kind, Tochter, Sohn, Bruder, Schwester, Großvater, Großmutter, Tante, Onkel, Cousin, Cousine.

LEHRSTUHL FÜR INFORMATIK III
Rheinisch-Westfälische Technische Hochschule Aachen
Univ.-Prof. Dr.-Ing. M. Nagl

Prof. Dr.-Ing. Manfred Nagl
Katja Cremer
Dirk Jäger
(*katja*|*jaeger*)*@i3.informatik.rwth-aachen.de*

Programmierung — WS 1999/2000 — 14. Übungsblatt

Ausgabe: Montag, 31.1.00 **Abgabe bis:** Montag, 7.2.00, 13.30 Uhr

Besprechung: in den Übungsgruppen am 10. und 11.2.00

Aufgabe 14.1 ***(7 Punkte)***
Das Spiel Nimm ist ein 2-Personen-Spiel, bei dem am Anfang eine Reihe von Streichhölzern auf dem Tisch liegt. Abwechselnd nimmt jeder Spieler höchstens 3 aber mindestens 1 Streichholz aus der Reihe weg. Gewonnen hat derjenige Spieler, der die letzten Streichhölzer aufnimmt.
Eine Anzahl Streichhölzer ist entweder eine Gewinnsituation oder eine Verlustsituation. Eine Gewinnsituation X ist dabei wie folgt definiert: Es gibt eine Anzahl i ($1 \le i \le 3$) von wegzunehmenden Streichhölzern, so daß der Gegner bei eigener optimaler Spielweise verliert. Eine Verlustsituation liegt vor, wenn X keine Gewinnsituation ist.
Schreiben Sie ein Prolog-Programm, das entscheidet, ob eine Spielsituation eine Gewinn- oder eine Verlustsituation ist. Das Programm soll Anfragen der Art `gewinn(`<*nummer*>`)` beantworten, wobei <*nummer*> die Zahl der vorhandenen Streichhölzer ist und das Programm mit ‚`yes`‘ genau dann antwortet, wenn der am Zug befindliche Spieler gewinnt (gewinnen kann bei optimaler Spielweise).
Folgende Sprachelemente von Prolog werden für die Lösung benötigt:

- der Operator `not`, dessen Argument zu klammern ist
- arithmetische Operatoren in der üblichen Infix-Notation
- Vergleiche in der üblichen Notation

Aufgabe 14.2 ***(5 Punkte)***
Welchen Wert berechnen die folgenden LISP-Ausdrücke:

- `(CAR (QUOTE (X Y Z)))`
- `(PLUS 7 (CAR (QUOTE (2 8))))`
- `(CDR (LIST (QUOTE (4 6)) 3))`
- `(CONS 3 (LIST 7 8 1))`
- `(CONS (QUOTE CDR) (LIST (QUOTE P)))`

Aufgabe 14.3 ***(8 Punkte)***

a) Die LISP-Funktion `DIVIDE` berechnet aus der Liste `(X Y)` von Argumenten die Liste (`X`*div*`Y` `X`*mod*`Y`). Definieren Sie eine Funktion, die aus den gleichen Argumenten eine Liste mit der umgekehrten Reihenfolge erzeugt, also (`X`*mod*`Y` `X`*div*`Y`). *(4 Punkte)*

b) Definieren Sie die rekursive LISP-Funktion `FIBON`, so daß `(FIBON N)` die n-te Fibonacci-Zahl berechnet.
(Hinweis: Die Fibonacci-Zahlen sind folgendermaßen definiert: $F_1 = 1$, $F_2 = 2$, $F_n = F_{n-1} + F_{n-2}$.)
(4 Punkte)