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

Übung 6
Musterlösung

Aufgabe 6.1 a), b) und c)

```
MODULE Matrizen EXPORTS Main;

(* Dieses Programm multipliziert zwei (5 x 5) Matrizen und
   gibt die Ergebnismatrix aus.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 23.11.00
   Letzte Aenderung: 23.11.00 *)

IMPORT SIO; (* Importiere notwendige Ein-/Ausgabeoperationen *)
IMPORT Fmt; (* Importiere Operationen, um Ausgabe zu formatieren *)

CONST N = 5; (* Konstante für den maximalen Index n *)

(* Notwendige Datentypen für eine (n x n) Matrix *)
TYPE Zeile  = [1 .. N];
     Spalte = [1 .. N];
     Matrix = ARRAY Zeile, Spalte OF REAL;

(* Deklaration der Matrizen/Initialisierung mit Feldaggregat *)
VAR a := Matrix {ARRAY Zeile OF REAL {1.0, 2.0, 3.0, 4.0, 5.0},
                 ARRAY Zeile OF REAL {1.5, 2.5, 3.5, 4.5, 5.5},
                 ARRAY Zeile OF REAL {1.3, 2.3, 3.3, 4.3, 5.3},
                 ARRAY Zeile OF REAL {1.7, 2.7, 3.7, 4.7, 5.7},
                 ARRAY Zeile OF REAL {1.9, 2.9, 3.9, 4.9, 5.9}};
    b := Matrix {ARRAY Zeile OF REAL {2.0, 3.0, 4.0, 5.0, 6.0},
                 ARRAY Zeile OF REAL {2.5, 3.5, 4.5, 5.5, 6.5},
                 ARRAY Zeile OF REAL {2.3, 3.3, 4.3, 5.3, 6.3},
                 ARRAY Zeile OF REAL {2.7, 3.7, 4.7, 5.7, 6.7},
                 ARRAY Zeile OF REAL {2.9, 3.9, 4.9, 5.9, 6.9}};

PROCEDURE MatrixMult(a, b: Matrix): Matrix =
(* Multipliziert die gegebenen Matrizen a und b, liefert die
   Ergebnismatrix als Rückgabewert *)
VAR c: Matrix;
BEGIN
  FOR i := FIRST(Zeile) TO LAST(Zeile) DO
    FOR j := FIRST(Spalte) TO LAST(Spalte) DO
      c [i, j] := 0.0;
      FOR k := FIRST(Spalte) TO LAST(Spalte) DO
        c [i, j] := c [i, j] + a[i, k] * b[k, j];
      END;
    END;
  END;
  RETURN c;
END MatrixMult;
```

```
PROCEDURE MatrixAusgeben(a: Matrix) =
(* Gibt die komplette Matrix elementweise aus *)
BEGIN
  FOR i := FIRST(Zeile) TO LAST(Zeile) DO
    FOR j := FIRST(Spalte) TO LAST(Spalte) DO
      SIO.PutText(MatElementFormat(a [i, j])); SIO.PutText("  ");
    END;
    SIO.Nl();
  END;
END MatrixAusgeben;

PROCEDURE MatElementFormat(r: REAL): TEXT =
(* Formatiere die REAL-Zahl mit Fixpunktnotation und 3 Nachkomma-
   stellen rechtsbuendig in einem festen Feld mit 8 Stellen *)
(* War in der Aufgabe NICHT verlangt! *)
BEGIN
  RETURN Fmt.Pad(Fmt.Real(r, Fmt.Style.Fix, 3),
                 8, ' ', Fmt.Align.Right);
END MatElementFormat;

BEGIN
  (* Ausgeben der zwei zu multiplizierenden Matrizen *)
  MatrixAusgeben(a);
  SIO.PutLine("                          *");
  MatrixAusgeben(b);
  SIO.PutLine("                          =");
  (* Ausgabe des Resultats *)
  MatrixAusgeben(MatrixMult(a, b));
END Matrizen.
```

Ergebnis des Programmlaufs:

```
   1.000     2.000     3.000     4.000     5.000
   1.500     2.500     3.500     4.500     5.500
   1.300     2.300     3.300     4.300     5.300
   1.700     2.700     3.700     4.700     5.700
   1.900     2.900     3.900     4.900     5.900
                        *
   2.000     3.000     4.000     5.000     6.000
   2.500     3.500     4.500     5.500     6.500
   2.300     3.300     4.300     5.300     6.300
   2.700     3.700     4.700     5.700     6.700
   2.900     3.900     4.900     5.900     6.900
                        =
  39.200    54.200    69.200    84.200    99.200
  45.400    62.900    80.400    97.900   115.400
  42.920    59.420    75.920    92.420   108.920
  47.880    66.380    84.880   103.380   121.880
  50.360    69.860    89.360   108.860   128.360
```

Aufgabe 6.2 a), b) und c)

```
MODULE Termine EXPORTS Main;

(* Dieses Programm realisiert eine simple Terminverwaltung, die
   zuvor eingegebene Termine sortiert ausgibt.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows NT 4.0
   Erstellt        : 23.11.00
   Letzte Aenderung: 23.11.00 *)

IMPORT SIO; (* Importiere notwendige Ein-/Ausgabeoperationen *)

CONST MAXTERMIN = 10; (* Maximale Anzahl zu verwaltender Termine *)

TYPE (* Unterbereichtstypen und Typ Uhrzeit fuer die Zeitangabe *)
     Stunden        = [0..23];

     Minuten        = [0..59];

     Uhrzeit        = RECORD
                        stunde: Stunden;
                        minute: Minuten;
                      END;

     (* Typ Termin fuer einzelnen Termin: Uhrzeit/Beschreibung *)
     Termin         = RECORD
                        beschreibung: TEXT;
                        zeit        : Uhrzeit;
                      END;

     (* Typ Terminkalender fuer maximal MAXTERMIN Termine *)
     Terminkalender = ARRAY [1..MAXTERMIN] OF Termin;


PROCEDURE TerminEingeben(): Termin =
(* Eingeben eines einzelnen Termins *)
VAR ergTermin: Termin;
BEGIN
  SIO.PutText("Beschreibung des Termins: ");
  ergTermin.beschreibung := SIO.GetLine();
  ergTermin.zeit  := UhrzeitEingeben();
  RETURN ergTermin;
END TerminEingeben;

PROCEDURE TerminAusgeben(termin: Termin) =
(* Ausgeben eines einzelnen Termins *)
BEGIN
  UhrzeitAusgeben(termin.zeit);
  SIO.PutText(" ");
  SIO.PutText(termin.beschreibung);
END TerminAusgeben;
```

```
PROCEDURE UhrzeitEingeben(): Uhrzeit =
(* Eingeben der Uhrzeit mit Stunden und Minuten *)
VAR ergUhrzeit: Uhrzeit;
    restZeile : TEXT;
BEGIN
  SIO.PutText("Uhrzeit (Stunden): ");
  ergUhrzeit.stunde := SIO.GetInt();
  (* Lesen der Restzeile mit RETURN *)
  restZeile := SIO.GetLine();

  SIO.PutText("Uhrzeit (Minuten): ");
  ergUhrzeit.minute := SIO.GetInt();
  (* Lesen der Restzeile mit RETURN *)
  restZeile := SIO.GetLine();

  RETURN ergUhrzeit;
END UhrzeitEingeben;

PROCEDURE StundenMinutenAusgeben(wert: [0..59]) =
(* Hilfsprozedur, um Stunden und Minuten zweistellig auszugeben *)
BEGIN
  IF wert < 10 THEN SIO.PutText("0"); END;
  SIO.PutInt(wert);
END StundenMinutenAusgeben;

PROCEDURE UhrzeitAusgeben(zeit: Uhrzeit) =
(* Gibt die Uhrzeit im Format Stunde:Minute aus *)
BEGIN
  StundenMinutenAusgeben(zeit.stunde);
  SIO.PutText(":");
  StundenMinutenAusgeben(zeit.minute);
  SIO.PutText(" Uhr");
END UhrzeitAusgeben;

PROCEDURE TerminkalenderAusgeben(termine: Terminkalender;
                           maxIndex: INTEGER; aufsteigend: BOOLEAN) =
(* Gibt den bis zum Eintrag maxIndex gefuellten Terminkalender
   termine abhaengig vom angegebenen BOOLEAN-Parameter aufsteigend
   oder absteigend aus. *)
VAR start, ende, schritt: INTEGER;

BEGIN
  IF aufsteigend THEN
    (* Initialisiere Schleifenwerte fuer aufsteigende Ausgabe *)
    start := FIRST(termine); ende := maxIndex; schritt := 1;
  ELSE
    (* Initialisiere Schleifenwerte fuer absteigende Ausgabe *)
    start := maxIndex; ende := FIRST(termine); schritt := -1;
  END;

  (* Ausgeben der Termine *)
  FOR i := start TO ende BY schritt DO
    TerminAusgeben(termine[i]); SIO.Nl();
  END;
END TerminkalenderAusgeben;
```

```
PROCEDURE TauscheTermin(VAR termin1, termin2: Termin) =
(* Hilfsprozedur, um einen einzelnen Termin zu tauschen *)
VAR speicherTermin: Termin;

BEGIN
  speicherTermin := termin1;
  termin1 := termin2;
  termin2 := speicherTermin;
END TauscheTermin;

PROCEDURE TerminkalenderSortieren(VAR termine: Terminkalender;
                                  maxIndex: INTEGER) =
(* Sortiert den bis maxIndex gefuellten Terminkalender termine in
   aufsteigende Reihenfolge *)
BEGIN
  (* Suche fuer jede Position im Terminkalender, ob sich auf den
     noch folgenden Positionen ein frueherer Termin findet *)
  FOR i := FIRST(termine) TO maxIndex DO
    FOR j := i + 1 TO maxIndex DO
      IF (termine[i].zeit.stunde > termine[j].zeit.stunde)  OR
         ( (termine[i].zeit.stunde = termine[j].zeit.stunde) AND
           (termine[i].zeit.minute > termine[j].zeit.minute) )
      THEN
        (* Tausche den Termin mit dem frueheren Termin *)
        TauscheTermin(termine[i], termine[j]);
      END;
    END;
  END;
END TerminkalenderSortieren;


VAR tKalender  : Terminkalender;
    terminIndex: [0..MAXTERMIN] := 0;

    (* Notwendig fuer die Benutzerinteraktion *)
    auswahl    : CHAR;
    restzeile  : TEXT;

BEGIN
  (* Benutzerinteraktion mittels REPEAT-Schleife *)
  REPEAT

    (* Auswahlmenue ausgeben *)
    SIO.Nl();
    SIO.PutLine("***  Terminkalender  ***");
    SIO.PutLine("(e) Neuen Termin eingeben");
    SIO.PutLine("(+) Aufsteigend sortiert ausgeben");
    SIO.PutLine("(-) Absteigend sortiert ausgeben");
    SIO.PutLine("(q) Beendet das Programm");
    SIO.Nl();

    (* Eingabe der ausgewaehlten Operation *)
    SIO.PutText("Auswahl: ");
    auswahl   := SIO.GetChar();
    restzeile := SIO.GetLine();
    SIO.Nl();
```

```
    (* Ausgewaehlte Operation ausfuehren *)
    CASE auswahl OF
       'e', 'E' => IF terminIndex < MAXTERMIN THEN
                     (* Solange noch Platz im Terminkalender *)
                     INC(terminIndex);
                     tKalender[terminIndex] :=TerminEingeben();

                     (* Terminkalender sofort sortieren *)
                     TerminkalenderSortieren(tKalender,
                                             terminIndex);
                   ELSE
                     SIO.PutLine("Terminkalender leider voll!");
                   END;

    |  '+'      => (* Terminkalender aufsteigend ausgeben *)
                   TerminkalenderAusgeben(tKalender,
                                          terminIndex, TRUE);
    |  '-'      => (* Terminkalender absteigend ausgeben *)
                   TerminkalenderAusgeben(tKalender,
                                          terminIndex, FALSE);
    |  'q', 'Q' => SIO.PutLine("Programmende!");
    ELSE
      SIO.PutLine("Ungueltige Eingabe!");
    END;
  UNTIL (auswahl = 'q') OR (auswahl = 'Q');
END Termine.
```

Probelauf des Programms für drei Termine:

```
***  Terminkalender  ***
(e) Neuen Termin eingeben
(+) Aufsteigend sortiert ausgeben
(-) Absteigend sortiert ausgeben
(q) Beendet das Programm

Auswahl: e

Beschreibung des Termins: Frühstücken
Uhrzeit (Stunden): 6
Uhrzeit (Minuten): 00

***  Terminkalender  ***
...

Auswahl: e

Beschreibung des Termins: Abendessen
Uhrzeit (Stunden): 20
Uhrzeit (Minuten): 30

***  Terminkalender  ***
...

Auswahl: e

Beschreibung des Termins: Mittagessen
Uhrzeit (Stunden): 13
Uhrzeit (Minuten): 00
```

```
***  Terminkalender  ***
...

Auswahl: +

06:00 Uhr Frühstücken
13:00 Uhr Mittagessen
20:30 Uhr Abendessen

***  Terminkalender  ***
...

Auswahl: -

20:30 Uhr Abendessen
13:00 Uhr Mittagessen
06:00 Uhr Frühstücken

***  Terminkalender  ***
...

Auswahl: q

Programmende!
```

Aufgabe 6.3 a), b), c) und d)

```
MODULE Meldeamt EXPORTS Main;

(* Dieses Programm realisiert eine einfache Verwaltung fuer
   Einwohnermeldedaten.

   Autor           : Moritz Schnizler, RWTH Aachen
   Umgebung        : PM 3, Windows 2000
   Erstellt        : 23.11.00
   Letzte Aenderung: 23.11.00 *)

IMPORT SIO;  (* Importiere notwendige Ein-/Ausgabeoperationen *)
IMPORT Text; (* Importiere notwendige Operationen für Typ TEXT *)

CONST MAXEINWOHNER = 10; (* Maximal zu verwaltende Einwohner *)

TYPE (* Typ Person umfasst alle Namen einer Person *)
     Person         = RECORD
                        familienname : TEXT;
                        rufname      : TEXT;
                        vorname      : ARRAY [2..5] OF TEXT;
                      END;

     (* Notwendige Unterbereichstypen und Typ fuer Tagesdatum *)
     Kalendertag    = [1..31];
     Kalendermonat  = [1..12];
     Kalenderjahr   = [1800..3000];   (* Ausreichender Zeitraum *)
```

```
    Tagesdatum     = RECORD
                       tag  : Kalendertag;
                       monat: Kalendermonat;
                       jahr : Kalenderjahr;
                     END;

    (* Typ Geburtsdaten besteht aus Geburtsort und -datum *)
    Geburtsdaten   = RECORD
                       ort  : TEXT;
                       datum: Tagesdatum;
                     END;

    (* Typ Geschlecht als Aufzaehlungstyp *)
    Geschlecht     = {maennlich, weiblich};

    (* Notwendige Unterbereichstypen und Typ fuer Adresse *)
    Hausnummer     = [1..1000];
    Postleitzahl   = [0..99999];

    Adresse        = RECORD
                       strasse: TEXT;
                       nummer : Hausnummer;
                       zusatz : TEXT;  (* Appartment, Block, ... *)
                       plz    : Postleitzahl;
                       ort    : TEXT;
                     END;

    (* Typ Familienstand als Aufzaehlungstyp *)
    Familienstand  = {ledig, verheiratet, geschieden, verwitwet};

    (* Typ Einwohner beschreibt einen Gemeldeten komplett *)
    Einwohner      = RECORD
                       person       : Person;
                       geboren      : Geburtsdaten;
                       geschlecht   : Geschlecht;
                       wohnung      : Adresse;
                       familienstand: Familienstand;
                       ehepartner   : Person;
                       anmeldung    : Tagesdatum;
                     END;

    (* Typ Melderegister umfasst MAXEINWOHNER  viele Einwohner *)
    Melderegister  = ARRAY [1..MAXEINWOHNER] OF Einwohner;

(* Prozeduren für die Ein- und Ausgabe der Datenstrukturelemente *)

PROCEDURE PersonEingeben(): Person =
(* Eingeben aller Namen einer Person *)
VAR ergPerson: Person;
    i        : INTEGER;
BEGIN
  SIO.PutText("Familienname: ");
  ergPerson.familienname := SIO.GetLine();
  SIO.PutText("Rufname: ");
  ergPerson.rufname := SIO.GetLine();
```

```
  (* Eingabe von bis zu vier weiteren Vornamen.
     Abbruch, falls vier Namen oder nichts eingegeben wurde. *)
  i := FIRST(ergPerson.vorname) - 1; (* Minus 1, zuerst Inkrement *)

  REPEAT
    INC(i); (* Eingabe des naechsten Vornamens *)

    (* Drucke den Text "i. Vorname: " *)
    SIO.PutText(Text.FromChar(VAL(ORD('0')+i,CHAR) )&". Vorname: ");
    ergPerson.vorname[i] :=SIO.GetLine();
  UNTIL (i = LAST(ergPerson.vorname)) OR
        Text.Equal(ergPerson.vorname[i], "");

  RETURN ergPerson;
END PersonEingeben;

PROCEDURE PersonAusgeben(p: Person) =
(* Ausgeben aller Namen einer Person *)
VAR i: INTEGER;
BEGIN
  SIO.PutText(p.familienname);
  SIO.PutText(", ");
  SIO.PutText(p.rufname);
  (* Weitere Vornamen ausgeben, solange maximale Anzahl
     nicht erreicht und kein leerer Name angetroffen wurde. *)
  i := FIRST(p.vorname);
  WHILE (i <= LAST(p.vorname)) AND
        NOT Text.Equal(p.vorname[i], "") DO
    SIO.PutText(" ");
    SIO.PutText(p.vorname[i]);
    INC(i);
  END;
END PersonAusgeben;

PROCEDURE DatumEingeben(): Tagesdatum =
(* Eingeben eines vollstaendigen Datums *)
VAR ergDatum: Tagesdatum;
BEGIN
  SIO.PutText("Tag: ");
  ergDatum.tag   := UBEingeben(FIRST(Kalendertag),
                               LAST(Kalendertag));
  SIO.PutText("Monat: ");
  ergDatum.monat := UBEingeben(FIRST(Kalendermonat),
                               LAST(Kalendermonat));
  SIO.PutText("Jahr: ");
  ergDatum.jahr  := UBEingeben(FIRST(Kalenderjahr),
                               LAST(Kalenderjahr));
  RETURN ergDatum;
END DatumEingeben;

PROCEDURE DatumAusgeben(dat: Tagesdatum) =
(* Ausgeben eines vollstaendigen Datums *)
BEGIN
  SIO.PutInt(dat.tag); SIO.PutText(".");
  SIO.PutInt(dat.monat); SIO.PutText(".");
  SIO.PutInt(dat.jahr);
END DatumAusgeben;
```

```
PROCEDURE GeburtsdatenEingeben(): Geburtsdaten =
(* Eingeben der kompletten Geburtsdaten *)
VAR ergGeburtsdaten: Geburtsdaten;
BEGIN
  SIO.PutText("Geburtsort: ");
  ergGeburtsdaten.ort   := SIO.GetLine();
  SIO.PutLine("Geburtsdatum: ");
  ergGeburtsdaten.datum := DatumEingeben();
  RETURN ergGeburtsdaten;
END GeburtsdatenEingeben;

PROCEDURE GeburtsdatenAusgeben(gd: Geburtsdaten) =
(* Ausgeben der kompletten Geburtsdaten *)
BEGIN
  SIO.PutText("Geboren: ");
  DatumAusgeben(gd.datum);
  SIO.PutText(", ");
  SIO.PutText(gd.ort);
END GeburtsdatenAusgeben;

PROCEDURE GeschlechtEingeben(): Geschlecht =
(* Eingeben des Geschlechts einer Person *)
VAR ergGeschlecht: Geschlecht := FIRST(Geschlecht);
    c            : CHAR;
    restzeile    : TEXT;
    eingabeOK    : BOOLEAN := FALSE;
BEGIN
  (* Fuehre Abfrage solange durch, bis korrekte Eingabe erfolgt *)
  REPEAT
    SIO.PutText("Geschlecht (m, w): ");
    c := SIO.GetChar();
    restzeile := SIO.GetLine(); (* Liest Restzeichen und RETURN *)
    CASE c OF
      'm', 'M' => ergGeschlecht := Geschlecht.maennlich;
                  eingabeOK := TRUE;
    | 'w', 'W' => ergGeschlecht := Geschlecht.weiblich;
                  eingabeOK := TRUE;
    ELSE
      SIO.PutLine("Falsche Eingabe!");
    END;
  UNTIL eingabeOK;
  RETURN ergGeschlecht;
END GeschlechtEingeben;

PROCEDURE GeschlechtAusgeben(g: Geschlecht) =
(* Ausgeben des Geschlechts einer Person *)
BEGIN
  CASE g OF
    Geschlecht.maennlich => SIO.PutText("maennlich");
  | Geschlecht.weiblich  => SIO.PutText("weiblich");
  END;
END GeschlechtAusgeben;

PROCEDURE AdresseEingeben(): Adresse =
(* Eingeben einer vollstaendigen Adresse *)
VAR ergAdresse: Adresse;
BEGIN
  SIO.PutText("Strasse: ");
```

```
  ergAdresse.strasse := SIO.GetLine();
  SIO.PutText("Hausnummer: ");
  ergAdresse.nummer := UBEingeben(FIRST(Hausnummer),
                                  LAST(Hausnummer));
  SIO.PutText("Zusatz: ");
  ergAdresse.zusatz := SIO.GetLine();
  SIO.PutText("Postleitzahl: ");
  ergAdresse.plz := UBEingeben(FIRST(Postleitzahl),
                               LAST(Postleitzahl));
  SIO.PutText("Ort: ");
  ergAdresse.ort := SIO.GetLine();
  RETURN ergAdresse;
END AdresseEingeben;

PROCEDURE AdresseAusgeben(adr: Adresse) =
(* Ausgeben einer vollstaendigen Adresse *)
BEGIN
  (* Strasse mit Hausnummer und Zusatz in einer Zeile *)
  SIO.PutText(adr.strasse); SIO.PutText(" ");
  SIO.PutInt(adr.nummer); SIO.PutText(" ");
  SIO.PutText(adr.zusatz); SIO.Nl();
  (* Postleitzahl mit Ort in neuer Zeile *)
  PostleitzahlAusgeben(adr.plz); SIO.PutText(" ");
  SIO.PutText(adr.ort);
END AdresseAusgeben;

PROCEDURE FamilienstandEingeben(): Familienstand =
(* Eingeben des Familienstands einer Person *)
VAR ergFamilienstand: Familienstand := FIRST(Familienstand);
    c               : CHAR;
    restzeile       : TEXT;
    eingabeOK       : BOOLEAN := FALSE;
BEGIN
  (* Fuehre Abfrage solange durch, bis korrekte Eingabe erfolgt *)
  REPEAT
    SIO.PutText("Familienstand (l, v, g, w): ");
    c := SIO.GetChar();
    restzeile := SIO.GetLine(); (* Liest Restzeichen und RETURN *)

    CASE c OF
      'l', 'L' => ergFamilienstand := Familienstand.ledig;
                  eingabeOK := TRUE;
    | 'v', 'V' => ergFamilienstand := Familienstand.verheiratet;
                  eingabeOK := TRUE;
    | 'g', 'G' => ergFamilienstand := Familienstand.geschieden;
                  eingabeOK := TRUE;
    | 'w', 'W' => ergFamilienstand := Familienstand.verwitwet;
                  eingabeOK := TRUE;
    ELSE
      SIO.PutLine("Falsche Eingabe!");
    END;
  UNTIL eingabeOK;

  RETURN ergFamilienstand;
END FamilienstandEingeben;
```

```
PROCEDURE FamilienstandAusgeben(famstand: Familienstand) =
(* Ausgeben des Familienstands einer Person *)
BEGIN
  CASE famstand OF
    Familienstand.ledig       => SIO.PutText("ledig");
  | Familienstand.verheiratet => SIO.PutText("verheiratet");
  | Familienstand.geschieden  => SIO.PutText("geschieden");
  | Familienstand.verwitwet   => SIO.PutText("verwitwet");
  END;
END FamilienstandAusgeben;

PROCEDURE EinwohnerEingeben(): Einwohner =
(* Komplett Eingeben eines gemeldeten Einwohners *)
VAR ergEinwohner: Einwohner;
BEGIN
  ergEinwohner.person        := PersonEingeben();
  ergEinwohner.geboren       := GeburtsdatenEingeben();
  ergEinwohner.geschlecht    := GeschlechtEingeben();
  ergEinwohner.wohnung       := AdresseEingeben();
  ergEinwohner.familienstand := FamilienstandEingeben();

  IF ergEinwohner.familienstand = Familienstand.verheiratet THEN
    SIO.PutLine("Ehepartner: ");
    ergEinwohner.ehepartner := PersonEingeben();
  END;
  SIO.PutLine("Anmeldung: ");
  ergEinwohner.anmeldung := DatumEingeben();
  RETURN ergEinwohner;
END EinwohnerEingeben;

PROCEDURE EinwohnerAusgeben(einwohner: Einwohner) =
(* Komplett Ausgeben eines gemeldeten Einwohners *)
BEGIN
  PersonAusgeben       (einwohner.person); SIO.Nl();
  GeburtsdatenAusgeben (einwohner.geboren); SIO.Nl();
  GeschlechtAusgeben   (einwohner.geschlecht); SIO.Nl();
  SIO.Nl();
  AdresseAusgeben      (einwohner.wohnung); SIO.Nl();
  SIO.Nl();
  (* Familienstand und, falls verheiratet, Ehepartner ausgeben *)
  FamilienstandAusgeben(einwohner.familienstand);
  IF einwohner.familienstand = Familienstand.verheiratet THEN
    SIO.PutText(" mit ");
    PersonAusgeben(einwohner.ehepartner);
  END;
  SIO.Nl();
  (* Datum der Anmeldung ausgeben *)
  SIO.PutText("Gemeldet seit ");
  DatumAusgeben(einwohner.anmeldung); SIO.Nl();
END EinwohnerAusgeben;

(* Hilfsprozeduren fuer die Ein- und Ausgabe der Daten *)

PROCEDURE UBEingeben(minimum, maximum: INTEGER): INTEGER =
(* Eingeben einer ganzen Zahl innerhalb des zulaessigen
Unterbereichs *)
VAR ergInteger: INTEGER;
    eingabeOK : BOOLEAN := FALSE;
```

```
BEGIN
  (* Fuehre Abfrage solange durch, bis korrekte Eingabe erfolgt *)
  REPEAT
    ergInteger := SIO.GetInt();
    restzeile  := SIO.GetLine(); (* Liest Restzeichen und RETURN *)
    IF (ergInteger >= minimum) AND (ergInteger <= maximum) THEN
      (* Wert im zulaessigen Bereich *)
      eingabeOK := TRUE;
    ELSE
      SIO.PutText("Wert ungueltig! Bitte nochmal eingeben: ");
    END;
  UNTIL eingabeOK;
  RETURN ergInteger;
END UBEingeben;

PROCEDURE PostleitzahlAusgeben(plz: [0..99999]) =
(* Hilfsprozedur, Postleitzahl immer fuenfstellig ausgeben *)
VAR praefix: TEXT;
BEGIN
  (* Zu druckendes Praefix bestimmen *)
  CASE plz OF
    0..9         => praefix := "0000";
  | 10..99       => praefix := "000";
  | 100..999     => praefix := "00";
  | 1000..9999   => praefix := "0";
  | 10000..99999 => praefix := "";
  END;
  (* Erst Praefix, dann eigentliche Zahl ausgeben *)
  SIO.PutText(praefix);
  SIO.PutInt(plz);
END PostleitzahlAusgeben;

PROCEDURE MelderegisterAusgeben(register: Melderegister;
                                maxIndex: INTEGER) =
(* Gibt bis zum Eintrag maxIndex gefuelltes Melderegister aus *)
BEGIN
  (* Ausgeben aller Einwohner *)
  FOR i := FIRST(register) TO maxIndex DO
    SIO.PutInt(i);
    SIO.PutLine(". Einwohner:");
    EinwohnerAusgeben(register[i]);
    SIO.Nl();
  END;
END MelderegisterAusgeben;

VAR register     : Melderegister;
    registerIndex: [0..MAXEINWOHNER] := 0;
    (* Notwendig fuer Benutzerinteraktion *)
    auswahl      : CHAR;
    restzeile    : TEXT;
BEGIN
  (* Benutzerinteraktion mittels REPEAT-Schleife *)
  REPEAT
    (* Auswahlmenue ausgeben *)
    SIO.Nl();
    SIO.PutLine("***  Einwohnermeldeamt  ***");
    SIO.PutLine("(e) Neuen Einwohner anlegen");
    SIO.PutLine("(a) Alle Einwohner ausgeben");
```

```
    SIO.PutLine("(q) Programm beenden");
    SIO.Nl();

    (* Eingabe der ausgewaehlten Operation *)
    SIO.PutText("Auswahl: ");
    auswahl   := SIO.GetChar();
    restzeile := SIO.GetLine();
    SIO.Nl();

    (* Ausgewaehlte Operation ausfuehren *)
    CASE auswahl OF
       'e', 'E' => IF registerIndex < MAXEINWOHNER THEN
                     (* Solange noch Platz im Meldeamt *)
                     INC(registerIndex);
                     register[registerIndex] := EinwohnerEingeben();
                   ELSE
                     SIO.PutLine("Leider kein Platz mehr frei!");
                   END;
    |  'a', 'A' => MelderegisterAusgeben(register, registerIndex);
    |  'q', 'Q' => SIO.PutLine("Programmende!");
    ELSE
      SIO.PutLine("Ungueltige Eingabe!");
    END;
  UNTIL (auswahl = 'q') OR (auswahl = 'Q');
END Meldeamt.
```

Probelauf des Programms für einen Einwohner:

```
***  Einwohnermeldeamt  ***
(e) Neuen Einwohner anlegen
(a) Alle Einwohner ausgeben
(q) Programm beenden

Auswahl: e

Familienname: Müller
Rufname: Willi
2. Vorname: Hans
3. Vorname:
Geburtsort: Bonn
Geburtsdatum:
Tag: 13
Monat: 2
Jahr: 1956
Geschlecht (m, w): maennlich
Strasse: Bahnstr.
Hausnummer: 4
Zusatz: a
Postleitzahl: 43267
Ort: Musterstadt
Familienstand (l, v, g, w): v
Ehepartner:
Familienname: Müller
Rufname: Erika
2. Vorname: Susanne
3. Vorname:
Anmeldung:
Tag: 15
```

```
Monat: 7
Jahr: 1982

***  Einwohnermeldeamt  ***
...

Auswahl: a

1. Einwohner:
Müller, Willi Hans
Geboren: 13.2.1956, Bonn
maennlich

Bahnstr. 4 a
43267 Musterstadt

verheiratet mit Müller, Erika Susanne
Gemeldet seit 15.7.1982


***  Einwohnermeldeamt  ***
...

Auswahl: q

Programmende!
```