Übungsaufgaben C/C++ (WS 03/04)

Klaus Wachtler

5. Januar 2005

Inhaltsverzeichnis

1 Allgemeines

Es besteht keine Anwesenheitspflicht, aber die vorgesehenen Übungen können natürlich dank meiner Anwesenheit dazu genutzt werden, Fragen schnell und einfach zu klären.

Ansonsten bin ich auch per Telefon (05191/70271, 0171/4553039) oder email (klaus@wachtler.de) ansprechbar.

Die sinnvolle Bearbeitung und fristgerechte Abgabe aller Übungen ist zwingende Voraussetzung für eine Klausurteilnahme. Da die Abgabe der Übungen nicht vom Prüfungsamt kontrolliert wird, ist eine Anmeldung zur Klausur beim PA nicht hinreichend, um an der Klausur teilnehmen zu können.

Aufgaben werden für die Donnerstagsgruppe veröffentlicht und müssen in der Regel spätestens nach zwei Wochen abgegeben sein, sodaß auch die Mittwochsgruppen noch über eine Woche Zeit haben.

Bei umfangreicheren Aufgaben kann die Bearbeitungszeit auch länger sein. Der Abgabetermin ist bei jeder Aufgabe angegeben.

Die Lösungen werden nicht mit Noten bewertet und gehen nicht in die Klausurnote ein, müssen aber Mindestanforderungen erfüllen, um anerkannt zu werden.

Anforderungen an die Übungen (und prinzipiell an erstellte Programme im allgemeinen, auch außerhalb dieser Übung):

- selbstständige Bearbeitung
- Wenn ggf. Vorgehensweise und/oder Quelltextpassagen aus anderen Quellen übernommen wurden, muß das deutlich aus der Dokumentation hervorgehen.

  Nicht dokumentiertes Abkupfern ist ein ko-Kriterium, auch wenn ich es erst einen Tag vor der Klausur bemerke oder sogar hinterher! Auch wenn eine Bewertung der Übungen nicht Teil der Klausurnote wird, ist eine Abgabe trotzdem Voraussetzung für die Klausur.

  Dadurch wird ein Täuschungsversuch hier ebenso gehandhabt wie in der Klausur; die Prüfungsordnung ist in diesem Punkt zu Recht ziemlich rüde (nicht bestanden, Verwirken der mdl. Ergänzungsprüfung etc.). Dieser Punkt ist also wirklich ernst zu nehmen, weil man damit sein Studium auch zügig beenden kann!

  Solange man nur kleinere Teile (also nicht die ganze Lösung) oder nur Ideen, Vorgehensweisen oder Hilfsfunktionen von anderswo übernimmt, ist das vollkommen legal – solange man es klar dokumentiert, und nicht ausdrücklich oder stillschweigend als eigene Erfindung verkauft.
- Die abgegebenen Programme müssen natürlich kompilierbar und lauffähig sein und soweit möglich den aktuellen Standards entsprechen (ANSI-C bzw. ISO-C89, ISO-C99, ISO-C++).
- Als Testsystem verwende ich den GNU-C-Compiler (gcc) bzw. GNU-C++ (g++).

Die abgegebenen Lösungen müssen damit fehlerfrei und auch mit der Option `-Wall` ohne Warnungen kompilierbar sein, sonst werden sie nicht anerkannt (ich kann natürlich nicht von 50 Kandidaten alle Quelltexte verbessern, sodaß sie erst übersetzt werden können).

Es ist also sehr zu empfehlen, daß jeder seine Übungen unter eine aktuellen Linux anfertigt oder zumindest zum Schluß damit testet.

Linux beißt nicht! Die Rechner in den PC-Pools können alle unter Linux betrieben werden; außerdem kann Linux natürlich kostenlos auf privaten Rechnern installiert werden oder sogar nur von CD laufen, ohne an einer bestehenden Installation etwas zu ändern (Knoppix). Es sollte also einem Informatikstudenten zumutbar sein, seine Übungen unter Linux zu erstellen, oder zumindest vor der Abgabe darauf zu testen.

Außer in begründeten Ausnahmefällen muß jedes Programm portabel sein, also nicht nur auf einem bestimmten Betriebssystem lauffähig sein, sondern mit den gängigen Compilern unter einem beliebigen System nutzbar sein.

Werden nicht portable Eigenheiten benutzt, dann ist das deutlich zu dokumentieren, und zu begründen.

- Auf eine brauchbare Fehlerbehandlung, sinnvolles Abfangen von Fehlern (auch bei anderen als den vorgegebenen Testdaten) ist wie bei allen anderen Programmen auch zu achten.

- klare Dokumentation (beispielsweise als Kommentar im Quelltext, falls es sich nur um eine Quelltext-Datei handelt, oder in einer kleinen README-Datei) mit folgenden Mindestinformationen:

  - Ersteller (Name, Matrikelnummer)
  - Aufgabenstellung, Vorgehensweise (falls es nicht offensichtlich ist: mit Begründung, warum dieser Weg gewählt wurde und warum andere Lösungswege verworfen wurden)
  - Datum Erstellung/letzte Änderung
  - Historie (wann wurden welche Änderungen vorgenommen)
  - eingehaltene bzw. bewußt verletzte Standards
  - Wie bekommt man das Beispiel zum Laufen, was muß wie kompiliert und gelinkt werden? Auf welchem/n System(en) wurde das Programm getestet? Dazu bitte die manuellen Aufrufe angeben (`gcc -Wall -std=c99`..., `g++ -Wall`... usw.).
  - Kommentare, die für ein schnelles Verstehen des Quelltextes nötig sind. Damit sind nicht so überflüssige Bemerkungen in der Art gemeint:

    ```
    int i; // dies ist eine Variable
    ```

Als Anregung der Anfang eines Quelltextes aus einem realen Beispiel:

```
//////////////////////////////////////////////////////////////////////
//
// Time-stamp: "(05.11.01 19:19) lesemdf.c [Klaus Wachtler (aw38)]"
//
//////////////////////////////////////////////////////////////////////
//
// liest eine MDF-Datei in den Speicher, und schreibt die Werte
// relativ planlos zur Standardausgabe oder in eine Matlab-Struktur.
```

```
//
///////////////////////////////////////////////////////////////////////
//
// Copyright:
/////////////
//
// (C) 2001 IAT GmbH/Klaus Wachtler
//
//      **********************************************
//    *             *                                *
//   *               *    ##       ###     ########  *
//  *                 *   ##      ## ##       ##     *
//  *  .-=.  .-===.   *   ##     ##   ##      ##     *
//  * /   ;)/     ;)  *   ##    #########     ##     *
//   *    //     //  *    ##   ##       ##    ##     *
//    *  //     //  *                                *
//      *******//*************************************
//            //
//           //           INGENIEURGESELLSCHAFT
//          //            FÜR  AGGREGATETECHNIK
//         //
//                        Hollerithallee 11
//                        D-30419 Hannover
//
//                        Tel. (+49)-511-279 491-0
//                        Fax: (+49)-511-279 491-23
//                        info@iat-gmbh.com
//
//                        http://www.iat-gmbh.com
//
//
///////////////////////////////////////////////////////////////////////
//
// Aufgabenstellung:
////////////////////
//
// Einlesen einer MDF-Datei (siehe [1]) und Ausgabe der
// darin abgelegten Meßwerte als ASCII-Datei beziehungsweise
// Weiterverwendung in Matlab.
//
///////////////////////////////////////////////////////////////////////
//
// Allgemeines:
///////////////
//
// Little endian, big endian:
// --------------------------
//
// Die Werte in den MDF-Dateien sind generell nach little endian
// abgelegt (Werte mit 2, 4, und 8 Byte werden also so abgelegt, daß
// an niedrigen Positionen die niederwertigen Bytes liegen, und an
// weiter hinten liegenden Positionen höherwertige Bytes).
// Wenn dieses Programm auf einem System mit Intel-Prozessor läuft,
// kann man diese Werte direkt binär in den Speicher einlesen.
// Auf Systemen mit big endian-Architektur (fast alle nicht-Intel)
// muß dagegen zwischen Einlesen und erster Verwendung genau einmal
// eine Konvertierung erfolgen, bei der die Bytes aller Werte mit
// mindestens zwei Bytes entsprechend gedreht werden.
```

```
//
// Für alle Bereiche einer MDF-Datei, die in [1] beschrieben sind,
// also IDBLOCK, HGBBLOCK, DGBLOCK, etc. bis CCBLOCK, existiert
// jeweils eine Funktion lesemdf...(), die genau einen solchen Block
// aus der Datei einliest (ggf. inklusive zusätzlicher Daten, wie
// bei TXBLOCK und CCBLOCK).
// Alle diese Funktionen konvertieren sofort alle betroffenen Werte.
//
// Davon unberührt sind erstmal die eigentlichen Meßdaten, die
// recordweise gelesen werden.
// Hier wird die Konvertierung zu big endian ggf. in der Funktion
// dWertAusRecord() beim Umwandeln in eine double erledigt (und zwar
// nicht destruktiv, so daß dWertAusRecord() auf denselben
// Speicherberech mehrfach angewendet werden kann).
//
//
// Teilweise ungetestet
// --------------------
//
// Das Programm ist bislang nur anhand weniger Beispieldateien überprüft
// und dementsprechend mit Vorsicht zu genießen; der Code ist
// teilweise noch nicht ein einziges Mal durchlaufen.
// Beispielsweise kommen mir die Umrechnungsfunktionen cc_...()
// ziemlich spanisch vor.
//
//
// Versionen
// ---------
//
// Aus diesem Quelltext können verschiedene Versionen generiert werden.
//
// Wenn beim Kompilieren LESEMDF_ASCII definiert ist, dann werden
// die gefundenen Daten zur Standardausgabe geschrieben.
// Es wird ein kleines main() gebaut, das die Konvertierung initiiert.
//
// Wenn LESEMDF_MATLAB definiert ist, dann werden die gefundenen Daten
// in eine MEX-Struktur eingebaut.
// es gibt hier kein Hauptprogramm, sondern eine mexFunction(),
// die für MATLAB als Einsprungspunkt in die DLL/shared object dient.
//
//////////////////////////////////////////////////////////////////////
//
// Geschichte:
/////////////
//
// Datum:     Name: Vers. Aenderung:
//
// 24.02.2001 kw    0.0   erster Entwurf (von awleer.cpp übernommen)
//                        (Parsen Struktur MDF-Datei)
//
// 24.02.2001 kw          Extrahieren der Werte aus dem Datenbereich
//
// 26.12.2001 kw          Einbau Big Endian
//
// 27.02.2001 kw          Big endian auf SGI (auf Rechner
//                        wspool2serv.inform.fh-hannover.de)
//                        getestet
//
```

```
// 28.02.2001 kw    1.0   Korrektur Umrechnung in phys. Werte
//
// 02.03.2001 kw          Korrektur parametr. Umrechnung
//
// 20.04.2001 kw          Umstellung auf Matlab
//
// 25.04.2001 kw          Erweiterung auf beliebige Bitzahlen (bis 32)
//                        für signed und unsigned; dabei signed nicht
//                        getestet!
//
// 22.05.2001 kw          Ausgaben wahlweise nicht nach stdout,
//                        sondern in Matlab-String
//
// 29.05.2001 kw    1.1   - Einbau von parse.c
//                        - Beschränkung des Einlesens auf
//                          Teile der MDF-Datei mit der Option
//                          'select(...)'
//                        - Unterbinden jeglichen Einlesens mit
//                          'nodata'
//
// 12.06.2001 kw    1.11  Korrektur Einlesen von Gleitkommazahlen,
//                        measure23.dat als ASCII auswertbar,
//                        aber Absturz in Matlab
//
// 22.06.2001 kw          Datensatzauswahl
//
// 23.06.2001 kw          Doku
//
// 05.11.2001 kw    1.2   Lesen von unsortiert geschriebenen Records
//                        implementiert (damit ist ASDMicroSample.dat
//                        auswertbar)
//
////////////////////////////////////////////////////////////////////
//
// Noch offene Punkte:
/////////////////////
//
// - Speicherfreigabe unvollständig
//
// - TODO abarbeiten
//
// - Alle Konvertierfunktionen außer "parametrisch linear" und
//   "Polynomfunktion 6 Parameter" sind
//   ungetestet mangels Beispieldateien.
//
// - Funktionen und globale Variablen soweit möglich statisch machen
//
// - bisher sind nur sortiert geschriebene Datenrecords möglich
//   (DGBLOCK.nRecordID ist (0)).
//   Für unsortierte brauche ich ein Beispiel.
//
// - signed int mit krummen Bitzahlen nicht getestet mangels
//   Beispieldateien
//
////////////////////////////////////////////////////////////////////
//
// Umgebung:
///////////
```

```
//
// ANSI-C++
// big endian, little endian werden selbstständig erkannt.
//
/////////////////////////////////////////////////////////////////////
//
// getestet:
///////////
//
// + Visual C++ 5.0 (W2000dt.)
//   cl lesemdf.c
//
// + cygwin 1.1.8, gcc 2.95.2-6 (W2000dt.)
//   gcc -Wall -O1 lesemdf.c -o lesemdf -lm
//
//
// + g++ 2.95.2 (Linux 2.4.0)
//   gcc -Wall -O1 lesemdf.c -o lesemdf -lm
//   gcc -Wall -DLESEMDF_ASCII -O3 lesemdf.c -lm -o lesemdf
//
//
/////////////////////////////////////////////////////////////////////
//
// Literatur:
////////////
//
// [1] Formatspezifikation MDF-Format (1.0, Schwotzer 10.3.2000)
//
// [2] Formatspezifikation MDF-Format (1.3, Schwotzer 22.1.2001)
//
// [3] What Every Computer Scientist Should Know About Floating
//     Point Arithmetic
//     David Goldberg
//     http://www.docs.sun.com
//
// [4] Lecture Notes On the status of IEEE Standard 754
//     Prof. W. Kahan
//     http://http.cs.berkeley.edu/%7Ewkahan/ieee754status/ieee754.ps
//
// [5] Die C++ Programmiersprache
//     Bjarne Stroustrup
//     4., aktualisierte Auflage
//     ISBN 3-8272-1660-X
//
// [6] 0 ist false, alles andere true
//     Ein Einführung in ANSI-C
//     Klaus Wachtler
//     Stand: Mai 2001
//     http://www.wachtler.de
//
// [7] "MATLAB Application Programming Interface Guide"
//     Mathworks Inc. (http://www.mathworks.com)
//
/////////////////////////////////////////////////////////////////////

...
```

Abgabe

- Die Übungen müssen spätestens zu dem Termin abgegeben werden, der bei der jeweiligen Aufgabe angegeben ist.

  Wenn möglich ist es aber sinnvoll, die Übung bereits in der Ausgabewoche zu bearbeiten, um den Anschluß an den Vorlesungsstoff zu behalten.

  Wenn fristgerecht abgegeben wurde, aber die Mindestanforderungen nicht ganz erfüllt sind (siehe oben), werde ich eine verbesserte Version nachfordern, soweit mir das sinnvoll erscheint.

  Sind die Mindestanforderungen aber eklatant verletzt (inhaltlich, oder durch falsche Angaben über die Herkunft beispielsweise), werden die Übungen nicht anerkannt.

  Weil das natürlich Folgen für die Klausurteilnahme hat, sollte man solchen Ärger tunlichst vermeiden.

  Wer den Termin nicht einhalten kann, aber trotzdem prinzipiell weitermachen will beziehungsweise die Klausr mitschreiben möchte, sollte micn umgehend ansprechen, um eine Regelung zu treffen.

- Die Abgabe erfolgt sinnvollerweise per email an `klaus@wachtler.de`. In der *Betreff:*-Zeile bitte den Namen, die Matrikelnummer und die Nummer oder den Titel der gelösten Aufgabe angeben; das spart mir Arbeit beim Abhaken.

- Bitte keine HTML-Mails schicken! Diese werden bei mir ungelesen entsorgt, weil sie üblicherweise nur Spam enthalten. HTML-Mails sind also so gut wie „nicht abgegeben“.

- Falls es sich um mehrere Dateien handelt, können die Dateien gern als ZIP-Datei (`*.zip`) oder `tar`-Datei (`*.tar`) oder `tar+gzip` (`*.tar.gz` oder `*.tgz`) gepackt werden.

  Andere Formate (RAR, JAR, ...) machen mir unnötig Arbeit; bitte nicht verwenden.

  Die Dateien bitte so packen, daß sie in ein Unterverzeichnis mit einem sinnigen Namen (beispielsweise mit der Info, die auch in der *Betreff:*-Zeile steht) entpackt werden, und nicht alle im aktuellen Arbeitsverzeichnis landen (sonst muß ich jeweils erst hineingucken, ein Verzeichnis erstellen, dahin wechseln, dort entpacken).

  Bei Datei- und Verzeichnisnamen bitte keine Leerzeichen verwenden.

- Bitte keine überflüssigen Dateien schicken.

  `*.o` oder ausführbare Programme sind nur unnötiger Ballast, ebenso wie Projektdateien einer Entwicklungsumgebung. Sinnvoll sind in der Regel nur die Quelltexte, die Dokumentation dazu, sowie bei Bedarf Eingabedateien für die Programme (Testdaten) oder ein makefile.

- Ich werde nicht jede eingeflogene Übung einzeln bestätigen, solange ich nichts zu meckern habe. Aber von Zeit zu Zeit stelle ich die bisher erhaltenen zusammen, und veröffentliche diese Liste. Dann bitte beschweren, wenn ich etwas übersehen haben sollte.

2 Nullstellensuche

Hier geht es in erster Linie darum, die sinnvolle Aufteilung eines Programms in Funktionen und die Parameterübergabe zu üben.

2.1 Aufgabe: Finden einer Nullstelle

Für die Funktion:

$$f(x) = x^2 - \ln(x) - 2$$

sollen mit je einem kleinen C-Programm die beiden Nullstellen mit Intervallschachtelung gefunden werden, die im Bereich $[0.13, 0.15]$ beziehungsweise $[1.5, 1.7]$ sowie mit dem Newton-Verfahren ausgehend von den Stellen 0.14 beziehungsweise 1.6.

Anmerkung: mit ln ist der natürliche Logarithmus gemeint. Im Numerikskript von Rene Klingenberg wird dafür die Schreibweise log verwendet.

Spätester Abgabetermin ist Mittwoch, der 27.10.2004. Empfohlen ist eine Bearbeitung bis 20.10.2004.

Hinweis: Um die Gleitkommafunktionen wie beispielsweise `log()` der `math`-Bibliothek in C zu nutzen, muß man erstens `math.h` mit `#include` einfügen, und zweitens auf unixähnlichen Systemen die Option `-lm` beim Aufruf des Linkers angeben, also insgesamt beispielsweise `gcc -Wall mein.c -lm -o mein` aufrufen.

2.2 Lösung: Finden einer Nullstelle

```
/* Time-stamp: "03.01.05 09:45 nullstellen.c klaus@wachtler.de"
 *
 * Musterlösung zur C-Übung 2:
 *
 * "Finden einer Nullstelle"
 *
 *  * Für die Funktion:
 *    f(x) = x^2 - ln(x) - 2
 *
 * sollen mit je einem kleinen C-Programm die beiden
 * Nullstellen mit Intervallschachtelung gefunden werden,
 * die im Bereich [0.13,0.15] beziehungsweise [1.5,1.7]
 * sowie mit dem Newton-Verfahren ausgehend von den Stellen
 * $0.14$ beziehungsweise $1.6$.
 *
 * Anmerkung: mit ln ist der natürliche Logarithmus
 * gemeint. Im Numerikskript von Rene Klingenberg wird dafür
 * die Schreibweise log verwendet (anstatt ln).
 *
 * ANSI-C; getestet gcc 3.3.4
```

```
 *
 * Kompilieren mit
 *     gcc -Wall nullstellen.c -lm -o nullstellen
 *
 * Datum:     Name:       Änderungen:
 *
 * 06.11.2004 K. Wachtler Als Musterlösung erstellt
 *
 * 03.01.2005 K. Wachtler In NullstelleNewton() in der for-Schleife
 *                        beim Initialisieren von neueNaeherung
 *                        x0 durch letzteNaeherung ersetzt
 *                        (Hinweis von Ingo Bente 02.01.2005)
 */

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <math.h>

/* Die Funktion NullstelleIntervallschachtelung() versucht,
 * eine Nullstelle der Funktion fkt im Intervall [a,b] zu
 * berechnen, und liefert einen Wert ungleich 0 zurück
 * (true) zurück, wenn das geklappt hat.
 *
 * Die Intervallschachtelung wird abgebrochen, wenn das
 * Restintervall (in dem die Nullstelle liegen muß) kleiner
 * als die übergebene Genauigkeit wird; als Resultat wird
 * dann die Mitte des Restintervalls geliefert.
 *
 * In diesem Fall wird die gefundene Nullstelle in
 * *p_nullstelle abgelegt.
 *
 * Die tatsächliche Nullstelle liegt dann sicher nicht mehr
 * als die übergebene Genauigkeit vom Rückgabewert entfernt.
 *
 * Im Fehlerfall wird 0 zurückgegeben, und *p_nullstelle
 * wird nicht verändert.
 *
 * Fehler können sein:
 *  - fkt(a) und fkt(b) haben gleiches Vorzeichen, und sind
 *    beide ungleich 0.0
 *  - die Genauigkeit ist 0.0 oder negativ
 *  - p_nullstelle ist NULL
 *
 * Wenn a nicht kleiner ist als b, dann ist das kein Fehler.
 *
 */

int NullstelleIntervallschachtelung( double (*fkt)( double x ),
                                     double  a,
                                     double  b,
                                     double  genauigkeit,
                                     double *p_nullstelle
                                     )
```

```
{
  /* Um die Funktion nicht unnötig oft aufrufen zu müssen,
   * werden die Funktionswert an den Intervallgrenzen nur
   * einmal berechnet:
   */
  double  fkt_a = fkt( a );
  double  fkt_b = fkt( b );

  /* Testen der Eingabeparameter:
   */
  if( !p_nullstelle
      ||
      genauigkeit<=0.0
      ||
      ( fkt_a>0.0 && fkt_b>0.0 )
      ||
      ( fkt_a<0.0 && fkt_b<0.0 )
      )
  {
    return 0; /* Fehler beim Aufruf! */
  }

  while( fabs( a - b )>=genauigkeit )
  {
    /* Wenn die Funktion in der Mitte des Intervalls das
     * selbe Vorzeichen hat wie am linken Rand, dann wird
     * die linke Grenze zur Mitte gezogen; ansonsten die
     * rechte:
     */
    double mitte = ( a + b )*0.5;
    double fkt_mitte = fkt( mitte );
    if( fkt_a>0.0 == fkt_mitte>0.0 )
    {
      a = mitte;
      fkt_a = fkt_mitte;
    }
    else
    {
      b = mitte;
      fkt_b = fkt_mitte;
    }
  }

  *p_nullstelle =  ( a + b )*0.5;
  return 1;
}


/* Die Funktion NullstelleNewton() versucht, eine Nullstelle
 * der Funktion fkt() in der Nähe von x0 zu berechnen, und
 * liefert einen Wert ungleich 0 zurück, wenn das geklappt
 * hat.  dfktdx() muß die Ableitung von fkt() nach dx für
 * ein übergebenes x liefern.
 *
```

```
 * Die Iteration wird abgebrochen, wenn die Änderung in
 * einem Schritt kleiner ist als die übergebene Genauigkeit.
 * In diesem Fall wird die gefundene Nullstelle in
 * *p_nullstelle abgelegt.
 *
 * Die tatsächliche Nullstelle kann auch weiter vom
 * Rückgabewert entfernt sein als die übergebene Genauigkeit
 * fordert, oder sogar überhaupt nicht existieren!
 *
 * Im Fehlerfall wird 0 zurückgegeben, und *p_nullstelle
 * nicht verändert.
 *
 * Fehler können sein:
 *  - die geforderte Genauigkeit ist nach iMax Schritten
 *    noch nicht erreicht
 *  - die Genauigkeit ist 0.0 oder negativ
 *  - p_nullstelle ist NULL
 *
 */
int NullstelleNewton( double (*fkt)( double x ),
                      double (*dfktdx)( double x ),
                      double  x0,
                      size_t  iMax,
                      double  genauigkeit,
                      double *p_nullstelle
                      )
{
  /* Testen der Eingabeparameter:
   */
  if( !p_nullstelle
      ||
      genauigkeit<=0.0
      )
  {
    return 0; /* Fehler beim Aufruf! */
  }


  /* Eingabe ok; Nullstelle berechnen:
   */
  size_t  iNaeherung;
  double  letzteNaeherung = x0;
  for( iNaeherung=0; iNaeherung<iMax; iNaeherung++ )
  {
    /* neue Näherung der Nullstelle:
     */
    double neueNaeherung
      = ( letzteNaeherung /* 03.01.2005 kw: korrigiert; bisher x0 */
          -
          fkt( letzteNaeherung )/dfktdx( letzteNaeherung )
          );

    //printf( "neueNaeherung = %f\n", neueNaeherung );
```

```
    /* Wenn der Abstand kleiner als genauigkeit wird, sind
     * wir fertig:
     */
    if( fabs( neueNaeherung - letzteNaeherung )<=genauigkeit )
    {
      *p_nullstelle = neueNaeherung;
      return 1;
    }

    /* Ansonsten für den nächsten Schritt die aktuelle
     * Näherung als die letzte merken:
     */
     letzteNaeherung = neueNaeherung;
  }

  /* Wenn wir hier ankommen, dann reichten iMax
   * Rechenschritte nicht.  Also mit Fehlermeldung
   * abbrechen:
   */
  return 0;
}


/* Die Funktion, deren Nullstelle gesucht wird:
 */
double f( double x )
{
  return x*x - log( x ) - 2.0;
}

/* Die Ableitung von f (nötig für das Newtonverfahren):
 */
double dfdx( double x )
{
  return 2.0*x - 1.0/x;
}

int main( int nargs, char **args )
{
  double nullstelle;

  if( NullstelleIntervallschachtelung( f,
                                       0.13,
                                       0.15,
                                       0.00001,
                                       &nullstelle
                                       )
      )
  {
    printf( "Intervallschachtelung: Nullstelle gefunden bei %f\n",
            nullstelle
            );
  }
  else
```

```
{
  printf( "Fehler bei Intervallschachtelung:"
          " keine Nullstelle gefunden!\n"
          );
}

if( NullstelleIntervallschachtelung( f,
                                     1.5,
                                     1.7,
                                     0.00001,
                                     &nullstelle
                                     )
    )
{
  printf( "Intervallschachtelung: Nullstelle gefunden bei %f\n",
          nullstelle
          );
}
else
{
  printf( "Fehler bei Intervallschachtelung:"
          " keine Nullstelle gefunden!\n"
          );
}

if( NullstelleNewton( f,
                      dfdx,
                      0.14,
                      100000,
                      0.01,
                      &nullstelle
                      )
    )
{
  printf( "Newton: Nullstelle gefunden bei %f\n",
          nullstelle
          );
}
else
{
  printf( "Fehler bei Newton: keine Nullstelle gefunden!\n" );
}

if( NullstelleNewton( f,
                      dfdx,
                      1.6,
                      100000,
                      0.001,
                      &nullstelle
                      )
    )
{
  printf( "Newton: Nullstelle gefunden bei %f\n", nullstelle );
}
```

```
  else
  {
    printf( "Fehler bei Newton: keine Nullstelle gefunden!\n" );
  }

  return 0;
} /* main( int nargs, char **args ) */
```

3 Strings

3.1 Aufgabe: Strings umgekehrt kopieren

Es sind die folgenden Funktionen zu schreiben und zu testen:

```
char *strrcpy( char *dest, const char *src );
char *strrcat( char *dest, const char *src );
char *strnrcpy( char *dest, const char *src, size_t n );
char *strnrcat( char *dest, const char *src, size_t n );
```

Diese sollen sich wie `strcpy()`, `strcat()`, `strncpy()` beziehungsweise `strncat()` verhalten (siehe `man 3 strcpy` und `man 3 strcat`). Allerdings sollen die kopierten Zeichen gerade in der entgegengesetzten Reihenfolge kopiert werden.

Also kopiert `strrcpy( puffer, "Emil" )` den Text `"limE"` in den `puffer`; die anderen drei Funktionen analog.

Spätester Abgabetermin ist Mittwoch, der 03.11.2004. Empfohlen ist eine Bearbeitung bis 27.10.2004.

3.2 Lösung: Strings umgekehrt kopieren

```
/* Time-stamp: "10.11.04 21:29 strr.c klaus@wachtler.de"
 *
 * Musterlösung zur Übungsaufgabe:
 * "Strings umgekehrt kopieren"
 *
 * Es sind die folgenden Funktionen zu schreiben und zu testen:
 *   char *strrcpy( char *dest, const char *src );
 *   char *strrcat( char *dest, const char *src );
 *   char *strnrcpy( char *dest, const char *src, size_t n );
 *   char *strnrcat( char *dest, const char *src, size_t n );
 *
 * Diese sollen sich wie strcpy(), strcat(), strncpy()
 * beziehungsweise  strncat() verhalten (siehe man 3 strcpy
 * und man 3 strcat). Allerdings sollen die kopierten Zeichen
 * gerade in der entgegengesetzten Reihenfolge kopiert werden.
 *
 * Also kopiert strrcpy( puffer, "Emil" ) den Text "limE" in
 * den puffer; die anderen drei Funktionen analog.
 *
 * Kompilieren mit
 *     gcc -Wall strr.c -o strr
 *
 * [1] man-Page (3) zu strcpy/strncpy
 *
 * [2] man-Page (3) zu strcat/strncat
```

```
 *
 * Datum:     Name:       Änderungen:
 * 06.11.2004 K. Wachtler Als Musterlösung erstellt
 *
 */

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>

/* kopiert den nullterminierten String ab src nach dest, und dreht
 * dabei die Reihenfolge der Zeichen um.
 * Der resultierende String in dest ist wieder nullterminiert.
 *
 * Rückgabewert ist der erste Parameter, also ein Zeiger auf die
 * neue Kopie.
 */
char *strrcpy( char *dest, const char *src )
{
  char       *dest_p = dest;
  const char *src_p  = src;

  /* zuerst mit src_p ans Ende des Quellstrings gehen:
   */
  while( *src_p )
  {
    src_p++;
  }
  /* Jetzt zeigt src_p auf die abschließende 0.
   */

  /* Jetzt wieder bis zum Anfang zurückgehen, und alle Zeichen
   * nach dest_p kopieren (dabei VOR der abschließenden 0
   * anfangen):
   */
  while( --src_p>=src )
  {
    *dest_p++ = *src_p;
  }

  /* src terminieren:
   */
  *dest_p = '\0';

  return dest;
}


/* kopiert den nullterminierten String ab src nach dest, und dreht
 * dabei die Reihenfolge der Zeichen um; allerdings werden maximal
 * n Zeichen kopiert.
 *
 * Der resultierende String in dest ist wieder nullterminiert,
```

```
 * falls n nicht erreicht wurde.
 *
 * Rückgabewert ist der erste Parameter, also ein Zeiger auf die
 * neue Kopie.
 */
char *strnrcpy( char *dest, const char *src, size_t n )
{
  char       *dest_p = dest;
  const char *src_p  = src;

  /* zuerst mit src_p ans Ende des Quellstrings gehen:
   */
  while( *src_p )
  {
    src_p++;
  }
  /* Jetzt zeigt src_p auf die abschließende 0.
   */

  /* Jetzt wieder bis zum Anfang zurückgehen, und alle Zeichen
   * nach dest_p kopieren (dabei VOR der abschließenden 0
   * anfangen):
   */
  while( --src_p>=src && dest_p<&dest[n] )
  {
    *dest_p++ = *src_p;
  }

  /* src terminieren:
   */
  if( dest_p<&dest[n] )
  {
    *dest_p = '\0';
  }

  return dest;
}


/* hängt an den nullterminierten String ab src den
 * nullterminierten String dest, und dreht dabei die Reihenfolge
 * der kopierten Zeichen um.
 *
 * Der resultierende String in dest ist wieder nullterminiert.
 *
 * Rückgabewert ist der erste Parameter, also ein Zeiger auf die
 * neue Kopie.
 */
char *strrcat( char *dest, const char *src )
{
  char       *dest_p = dest;

  /* Zuerst das Ende von dest finden:
   */
```

```
  while( *dest_p )
  {
    dest_p++;
  }

  /* Ab hier einfach drankopieren:
   */
  strrcpy( dest_p, src );
  return dest;
}

/* hängt an den nullterminierten String ab src den
 * nullterminierten String dest, und dreht dabei die Reihenfolge
 * der kopierten Zeichen um; allerdings werden maximal n Zeichen
 * kopiert.
 *
 * Der resultierende String in dest ist wieder nullterminiert,
 * falls n nicht erreicht wurde.
 *
 * Rückgabewert ist der erste Parameter, also ein Zeiger auf die
 * neue Kopie.
 */
char *strnrcat( char *dest, const char *src, size_t n )
{
  char       *dest_p = dest;


  /* Zuerst das Ende von dest finden:
   */
  while( *dest_p )
  {
    dest_p++;
  }

  /* Ab hier einfach drankopieren:
   */
  strnrcpy( dest_p, src, n );
  return dest;
}


int main( int nargs, char **args )
{
  char puffer[100];

  strrcpy( puffer, "Ich bin doof" );
  printf( "Nach strrcpy(): <%s>\n", puffer );

  strrcat( puffer, "Der Hund ist schlau. " );
  printf( "Nach strrcat(): <%s>\n", puffer );

  strnrcpy( puffer, "wenn das mal gutgeht!", 50 );
  printf( "Nach strnrcpy(): <%s>\n", puffer );
```

```
  strnrcat( puffer, "der Emir sagt: ", 50 );
  printf( "Nach strnrcat(): <%s>\n", puffer );

  strnrcpy( puffer, "Das kann nicht gut gehen, oder?", 5 );
  printf( "Nach strnrcpy(): <%s>\n", puffer );

  puffer[5] = 'n';
  strnrcat( puffer, "Ich meine:  ", 5 );
  printf( "Nach strnrcat(): <%s>\n", puffer );


  return 0;
} /* main( int nargs, char **args ) */
```

4 Einfach verkettete Liste

Sinn dieser Übung ist es, den Umgang mit Zeigern und mit freiem Speicher zu üben (`malloc()`, `free()`).

4.1 Aufgabe: Einfach verkettete Liste

Es ist ein Datentyp für eine Person zu definieren (`struct`) mit mindestens Vor- und Nachname und Alter.

Daraus soll eine einfach verkettete Liste gebaut werden, indem jeweils zum

- Einfügen einer Person am Anfang
- Einfügen am Ende
- Suche nach dem Nachnamen
- Löschen eines Elements anhand des Nachnamens
- Iteration über alle Elemente und Anwenden einer (als Zeiger übergebenen) Funktion auf jedes Listenelement
- Freigeben aller Elemente einer Liste

je eine Funktion geschrieben wird.

Die Funktionen sollen so entworfen werden, daß sie prinzipiell mit mehreren gleichzeitig existierenden Listen ungehen können. Deshalb reicht es nicht, den Listenanfang in einer globalen Variablen zu halten. Vielmehr soll der Aufrufer den Listenanfang (oder bei Bedarf mehrere Listenanfänge) anlegen können, und dieser Listenanfang geeignet als Parameter übergeben werden. Funktionen, welche die Liste nicht manipulieren (Suche, Iterieren), kann dieser Listenanfang direkt übergeben werden; den manipulierenden Funktionen dagegen (Einfügen etc.) muß die Adresse der Variablen übergeben werden, die den Listenanfang hält.

Es ist darauf zu achten, daß nicht mehr benötigter Speicher sauber freigegeben wird.

Im Hauptprogramm werden diese Funktionen aufgerufen und mit verschiedenen Fällen (leere oder nicht leere Listen beispielsweise) getestet.

Spätester Abgabetermin ist Mittwoch, der 10.11.2004. Empfohlen ist eine Bearbeitung bis 03.11.2004.

4.2 Lösung: Einfach verkettete Liste

```
/* Time-stamp: "03.01.05 09:50 einfacheliste.c klaus@wachtler.de"
```

```
 *
 * Musterlösung zur C-Übung 4:
 *
 * "Einfach verkettete Liste"
 *
 * Es ist ein Datentyp für eine Person zu definieren (struct)
 * mit mindestens Vor- und Nachname und Alter.
 *
 * Daraus soll eine einfach verkettete Liste gebaut werden, indem
 * jeweils zum
 *  -  Einfügen einer Person am Anfang
 *  -  Einfügen am Ende
 *  -  Suche nach dem Nachnamen
 *  -  Löschen eines Elements anhand des Nachnamens
 *  -  Iteration über alle Elemente und Anwenden einer (als Zeiger
 *     übergebenen) Funktion auf jedes Listenelement
 *  -  Freigeben aller Elemente einer Liste
 * je eine Funktion geschrieben wird.
 *
 * Die Funktionen sollen so entworfen werden, daß sie prinzipiell
 * mit mehreren gleichzeitig existierenden Listen ungehen
 * können. Deshalb reicht es nicht, den Listenanfang in einer
 * globalen Variablen zu halten. Vielmehr soll der Aufrufer den
 * Listenanfang (oder bei Bedarf mehrere Listenanfänge) anlegen
 * können, und dieser Listenanfang geeignet als Parameter
 * übergeben werden. Funktionen, welche die Liste nicht
 * manipulieren (Suche, Iterieren), kann dieser Listenanfang
 * direkt übergeben werden; den manipulierenden Funktionen dagegen
 * (Einfügen etc.) muß die Adresse der Variablen übergeben werden,
 * die den Listenanfang hält.
 *
 * Es ist darauf zu achten, daß nicht mehr benötigter Speicher
 * sauber freigegeben wird.
 *
 * Im Hauptprogramm werden diese Funktionen aufgerufen und mit
 * verschiedenen Fällen (leere oder nicht leere Listen
 * beispielsweise) getestet.
 *
 * Datum:     Name:       Änderungen:
 *
 * 08.11.2004 K. Wachtler Als Musterlösung erstellt
 *
 * 10.11.2004 K. Wachtler Quelltext auf 66 Spalten umformatiert
 *
 * 03.01.2005 K. Wachtler In ListePerson_free Zeiger auf
 *                        freigegebenen Speicher zu NULL gesetzt
 *                        (Hinweis Ingo Bente 02.01.2005)
 *
 */

#include <stdlib.h>
#include <stddef.h>
#include <stdio.h>
#include <string.h>
```

```
/* Datentyp für eine Person:
 */
typedef struct
{
  char vname[40];
  char nname[40];
  int alter;
} person_t;

/* Datentyp für ein Listenelement:
 */
typedef struct structLEPerson
{
  person_t               person;
  struct structLEPerson *next;
} LEPErson_t;

void ListePerson_insert_vorne( LEPErson_t **lanf_p,
                               const char  *vn,
                               const char  *nn,
                               int          alter
                               )
{
  /* neues Element allokieren:
   */
  LEPErson_t *neuesElement
    = (LEPErson_t*)malloc( sizeof(LEPErson_t) );
  if( !neuesElement )
  {
    fprintf( stderr,
             "kein Speicher in ListePerson_insert_vorne()\n"
             );
    exit( 2 );
  }

  /* neues Element füttern:
   */
  strncpy( neuesElement->person.vname,
           vn,
           sizeof(neuesElement->person.vname)-1
           );
  neuesElement->person.vname[sizeof(neuesElement->person.vname)-1]
    = '\0';
  strncpy( neuesElement->person.nname,
           nn,
           sizeof(neuesElement->person.nname)-1
           );
  neuesElement->person.nname[sizeof(neuesElement->person.nname)-1]
    = '\0';
  neuesElement->person.alter = alter;
  neuesElement->next         = *lanf_p; /* Alter LA
                                         * wird Nachfolger
                                         */
```

```
  /* Neues Element als Listenanfang speichern:
   */
  *lanf_p = neuesElement;
}

void ListePerson_insert_hinten( LEPErson_t **lanf_p,
                                const char  *vn,
                                const char  *nn,
                                int          alter
                                )
{
  /* Zeiger auf LA durch die Liste laufen lassen, bis ein
   * next gefunden wird, welches NULL ist:
   */
  while( *lanf_p )
  {
    lanf_p = &((*lanf_p)->next);
  }
  /* Wenn es Elemente gibt, dann zeigt lanf_p jetzt auf das letzte
   * next der Liste (also das mit dem Wert NULL).
   *
   * Wenn die Liste leer ist, zeigt lanf_p auf den Listenanfang
   * wie vom Aufrufer in lanf_p übergeben.
   */

  /* Wenn wir jetzt so tun, als ob lanf_p der Listenanfang wäre,
   * können wir ListePerson_insert_vorne() zum Anfügen verwenden:
   */
  ListePerson_insert_vorne( lanf_p, vn, nn, alter );
}

/* Iteriert über alle Elemente, ruft für jedes Element die call
 * back-Funktion auf, und liefert die Anzahl der besuchten
 * Elemente zurück.
 */
int ListePerson_iteriere( LEPErson_t *liste,
                          void(*callback)( const person_t* )
                          )
{
  int anzahl = 0;
  while( liste )
  {
    callback( &(liste->person) );
    anzahl++;
    liste = liste->next;
  }
  return anzahl;
}


/* Löscht aus einer Liste alle Einträge, die den angegebenen
 * Nachnamen haben:
 */
```

```
void ListePerson_loesche_nn( LEPErson_t **lanf_p, const char *nn )
{
  /* alle Einträge abklappern, und mit dem übergebenen Namen
   * vergleichen:
   */
  while( *lanf_p )
  {
    if( !strcmp( (*lanf_p)->person.nname, nn ) )
    {
      /* Nachname gleich, also löschen:
       */
      /* Rest der Liste merken */
      LEPErson_t *alter_next = (*lanf_p)->next;
      free( *lanf_p );        /* Listenelement freigeben */
      *lanf_p = alter_next;   /* Rest der Liste
                               * wieder einhängen
                               */
    }
    else
    {
      /* aktuelles Element nicht zu löschen, also eins
       * weitergehen:
       */
      lanf_p = &((*lanf_p)->next);
    }
  }
}

/* Sucht anhand eines Nachnamens nach einer PErson und liefert
 * einen Zeiger darauf:
 */
const person_t *ListePerson_suche_nn( LEPErson_t *lanf,
                                      const char *nn
                                      )
{
  while( lanf )
  {
    if( !strcmp( lanf->person.nname, nn ) )
    {
      /* Element gefunden!
       */
      return &(lanf->person);
    }

    /* eins weiter gehen:
     */
    lanf = lanf->next;
  }
  /* nix gefunden, also NULL liefern:
   */
  return NULL;
}

/* Gibt eine Liste frei (und setzt den Listenanfang zu NULL):
```

```
 */
ListePerson_free( LEPErson_t **lanf_p )
{
  if( *lanf_p )
  {
    ListePerson_free( &((*lanf_p)->next) );
    free( *lanf_p );
    *lanf_p = NULL; /* eingefügt K. Wachtler 03.01.2005 */
  }
}

/*+++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

/* Testprogramm:
 */


/* Gib eine Person aus:
 */
void zeigeLEPerson( const person_t *pers_p )
{
  printf( "%s %s ist %d Jahre alt\n",
          pers_p->vname,
          pers_p->nname,
          pers_p->alter
          );
}

int main( int nargs, char **args )
{
  LEPErson_t   *liste1 = NULL;
  LEPErson_t   *liste2 = NULL;

  const person_t     *gefundene_person;

  /* Liste aufbauen:
   */
  ListePerson_insert_vorne( &liste1, "Klaus", "Wachtler", 42 );
  ListePerson_insert_vorne( &liste1, "Otto", "Waalkes", 58 );
  ListePerson_insert_vorne( &liste1, "Inge", "Meisel", 128 );
  ListePerson_insert_hinten( &liste1, "Angie", "Merkel", 72 );
  ListePerson_insert_hinten( &liste1, "Zorro", "Schatten", 22 );
  ListePerson_insert_hinten( &liste1, "Dschingis", "Khan", 321 );

  /* Liste ausgeben:
   */
  printf( "erste Liste:\n" );
  ListePerson_iteriere( liste1, zeigeLEPerson );

  /* Zweite Liste aufbauen:
   */
  printf( "zweite Liste:\n" );
  ListePerson_insert_hinten( &liste2, "Frieda", "Meier", 45 );
  ListePerson_insert_vorne( &liste2,
```

```
                             "Karl",
                             "Müüüüüüüüüüüüüüüüüüüüüüüüüüüüüüüller",
                             34
                             );
  ListePerson_insert_vorne( &liste2, "Esmeralda", "Meier", 23 );
  ListePerson_insert_vorne( &liste2, "Zumpfi", "Dumpfbacke", 12 );

  /* Liste ausgeben:
   */
  ListePerson_iteriere( liste2, zeigeLEPerson );

  printf( "nach Löschen:\n" );
  ListePerson_loesche_nn( &liste2, "Meier" );
  ListePerson_loesche_nn( &liste2, "Dumpfbacke" );
  ListePerson_iteriere( liste2, zeigeLEPerson );

  gefundene_person = ListePerson_suche_nn( liste1, "Waalkes" );
  if( gefundene_person )
  {
    printf( "Gefunden: %s %s, %d\n",
            gefundene_person->vname,
            gefundene_person->nname,
            gefundene_person->alter
            );
  }
  else
  {
    printf( "nichts gefunden!\n" );
  }

  /* Speicher freigeben:
   */
  ListePerson_free( &liste1 );
  ListePerson_free( &liste2 );
  return 0;
} /* main( int nargs, char **args ) */
```

5 Dateien

5.1 Aufgabe: Dateien

Die Aufgabe besteht aus drei Teilen:

1. Das zu schreibende Programm definiert einen Datentyp für Personen (`Vorname`, `Nachname`, `Alter`, `Hobby`), und fragt den Benutzer nach einer gewünschten Anzahl einzulesender Personen.

   Anschließend wird Speicher für entsprechend viele Personen allokiert, in einer Schleife werden die Daten aller Personen eingelesen, und danach in einer weiteren Schleife zur Standardausgabe ausgegeben.

   Der Einfachheit halber werden die Zeichenketten direkt in der `struct` mit einer festen Länge vereinbart (z.B. `char Vorname[40];`).

   Das Einlesen der Daten für die Personen kann mit `scanf()` mit der Formatangabe `%s` beziehungsweise `%d` erfolgen; in den Strings dürfen demnach keine Leerzeichen vorkommen (innerhalb eines Namens etwa). Alternativ wäre auch ein Einlesen einer ganzen Zeile mit `fgets()` möglich, und im Programm wird dann der Name von führenden oder schließenden Leerzeichen sowie einem eventuellen Zeilenvorschub bereinigt.

2. Es sind zwei Programme zu schreiben:

   (a) das erste Programm verhält sich wie in der vorhergehenden Teilaufgabe; allerdings werden die gelesenen Daten nicht zur Standardausgabe geschrieben, sondern stattdessen in eine vom Benutzer angegebene Textdatei in einem passend gewählten Format.

   (b) ein weiteres Programm fragt ebenfalls nach einem Dateinamen, öffnet mit diesem Namen die Datei und versucht alle Personen zu lesen (zur Kontrolle werden die gelesenen Daten zur Standardausgabe geschrieben).

   Was passiert, wenn in den Vor- oder Nachnamen Leerzeichen stehen?

3. Die Programme der vorhergehenden Teilaufgabe sollen so abgeändert werden, daß anstatt einer Textdatei eine binär geschriebene Datei verwendet wird.

   Was passiert jetzt mit Leerzeichen?

Bitte darauf achten, daß die Programme durch die eingelesenen Daten nicht abstürzen können, sondern auf unsinnige Eingaben kontrolliert mit einer Fehlermeldung und gegebenenfalls einem Programmabbruch reagieren.

Spätester Abgabetermin ist Mittwoch, der 17.11.2004. Empfohlen ist eine Bearbeitung bis 10.11.2004.

5.2 Lösung: Dateien

6 Binärgefummel (optional)

6.1 Aufgabe: Lesen von Binärdaten auf verschiedenen Systemen

Es ist ein Programm zu erstellen, welches Binärdateien liest, die unter Sun (big endian) oder Intel (little endian) erzeugt wurden, und den Inhalt der Dateien in Text umwandelt.

Zum Testen werden einige solche Dateien zur Verfügung gestellt, und zwar unter `http://www.wachtler.de/fhai/uebung6_bin_testdaten.zip` beziehungsweise auf dem Skripteserver in
`/home/daten/skripte/c-c++_wachtler/uebung6_bin_testdaten.zip`.

Zur Vereinfachung soll angenommen werden, daß sowohl in den Dateien als auch auf dem System des laufenden Programms der ASCII-Zeichensatz verwendet wird.

Als Datenformate kommen in den Dateien vor:

`STRING` (nullterminierte Zeichenkette, jeweils nur 7 Bit verwendet)

`SBYTE` (ganze Zahl, `signed`, 1 Byte)

`UBYTE` (ganze Zahl, `unsigned`, 1 Byte)

`SWORD` (ganze Zahl, `signed`, 2 Byte)

`UWORD` (ganze Zahl, `unsigned`, 2 Byte)

`SLONG` (ganze Zahl, `signed`, 4 Byte)

`ULONG` (ganze Zahl, `unsigned`, 4 Byte)

`FLOAT` (Gleitkommazahl, *single precision* mit 4 Byte gemäß IEEE754)

`DOUBLE` (Gleitkommazahl, *double precision* mit 8 Byte gemäß IEEE754)

Die Dateien haben eine Kennung in den ersten Byte:
`'U', 'e', 'b', 'u', 'n', 'g', ' ', 'C', '/', 'C', '+', '+', ' ', 'B', 'E'`, 0x00
(falls im BE-Format geschrieben)
`'U', 'e', 'b', 'u', 'n', 'g', ' ', 'C', '/', 'C', '+', '+', ' ', 'L', 'E'`, 0x00
(falls im LE-Format geschrieben)

Im weiteren Verlauf sind dann Einzelwerte beschrieben, oder Verläufe. Jeder Wert oder Verlauf wird durch eine zwei Byte große Kennung eingeleitet, dann kommt ein beliebig langer Name (terminiert mit einem Nullbyte) gefolgt von weiteren Daten.

6.1.1 Einzelwerte:

Kennung:	weitere Daten:
`0x0000`	*Name (*`STRING`*), Wert (*`STRING`*)*
`0x0001`	*Name (*`STRING`*), Wert (*`SBYTE`*)*
`0x0002`	*Name (*`STRING`*), Wert (*`UBYTE`*)*
`0x0003`	*Name (*`STRING`*), Wert (*`SWORD`*)*
`0x0004`	*Name (*`STRING`*), Wert (*`UWORD`*)*
`0x0005`	*Name (*`STRING`*), Wert (*`SLONG`*)*
`0x0006`	*Name (*`STRING`*), Wert (*`ULONG`*)*
`0x0007`	*Name (*`STRING`*), Wert (*`FLOAT`*)*
`0x0008`	*Name (*`STRING`*), Wert (*`DOUBLE`*)*

6.1.2 Beispielausgaben des Programms:

```
Variable (0x0007, FLOAT)
pi=3.141592

Variable (0x0007, DOUBLE)
klein=-1.678938e-12

Variable (0x0000, STRING)
ichbin="klaus"
```

Hinweis: Es wäre praktisch, wenn in ausgegebenen Strings Anführungsstriche (`"`) durch `\"` und alle nicht druckbaren Zeichen durch eine entsprechende andere Sequenz (`\t`, `\n`, `\007`) dargestellt werden.

6.1.3 Eindimensionale Felder:

Tabellen oder Felder mit einer Dimension haben ebenfalls einen Namen, sowie eine bestimmte Anzahl (n) Wertepaare. Als Datentyp des ersten Werts ebenso wie des zweiten Werts sind alle Typen der einfachen Variablen erlaubt (s.o.).

Das obere Byte der Kennung ist `0x10` (also ist die Kennung `0x10??`).

Im unteren Byte steht das obere Nibble für den Datentyp der ersten Spalte (also `0x101?` für `SBYTE`, `0x102?` für `UBYTE` etc.); analog steht das unterste Nibble für den Typ der zweiten Spalte. Beispielsweise steht die Kennung `0x1047` für eine Tabelle mit Wertepaaren, deren jeweils erster Wert ein UWORD ist (wegen der `4`), der zweite Wert jeden Wertepaares ist eine FLOAT (wegen der `7`).

Nach der Kennung folgt als `ULONG` die Anzahl n der Wertepaare, anschließend der Name des Feldes als nullterminierter String, dann für das erste Wertepaar der erste und der zweite Wert, gefolgt von den restlichen Wertepaaren.

Beispielausgaben des Programms:

```
Tabelle (0x1047, 3*UWORD/FLOAT)
"einetabelle"
   10  3.141593
   20  7.234455
   25 -1.488357e28

Tabelle (0x1000, , 4*STRING/STRING)
"eineanderetabelle"
   "Strauß"  "tot"
   "Kohl"    "weg"
   "Stoiber" "auch weg"
   "Merkel"  "kommt noch"
```

6.1.4 Zweidimensionale Felder:

Tabellen oder Felder mit zwei Dimensionen haben ebenfalls einen Namen, sowie eine bestimmte Anzahl (nx) x-Werte, eine Anzahl (ny) y-Werte, und (nx*ny) Funktionswerte. Als Datentyp der x-, y- und der Funktionswerte sind alle Typen der einfachen Variablen erlaubt (s.o.).

Das obere Nibble des oberen Bytes der Kennung ist `0x2` (also ist die Kennung `0x2???`).

Das untere Nibble des oberen Bytes gibt den Typ der x-Werte an (wie oben), also `0x20??` für `STRING`, `0x21??` für `SBYTE`, und so weiter.

Im unteren Byte steht das obere Nibble für den Datentyp der y-Werte (also `0x2?0?` für `STRING`, `0x2?1?` für `SBYTE` etc.).

Analog steht das unterste Nibble für den Typ der Funktionswerte.

Beispielsweise steht die Kennung `0x2158` für eine zweidimensionale Tabelle, deren x-Werte vom Typ `SBYTE` sind (`1`), die y-Werte vom Typ `SLONG` (`5`), und die Funktionswerte vom Typ `DOUBLE` (wegen der `8`).

Nach der Kennung folgt als `ULONG` die Anzahl der x-Werte (nx), dann ebenfalls als `ULONG` die Anzahl der y-Werte (ny),, der Name des Feldes als `STRING`, dann (nx) x-Werte, dann (ny) y-Werte, und letztlich (nx*ny) Funktionswerte in der Reihenfolge $f(x_1, y_1)$, $f(x_1, y_2)$, $\ldots$ $f(x_{nx}, y_{ny})$.

6.1.5 Beispielausgabe des Programms:

```
zweidimensionale Tabelle (0x2787, x=4*FLOAT, y=5*DOUBLE, Werte=FLOAT)
"nocheinetabelle"
   x/y   1.0   2.0    2.5     2.6    2.8
   1.0   3.0   3.256  3.533   3.3    3.2
   1.2   3.1   3.356  3.633   3.4    3.3
   1.4   3.2   3.456  3.733   3.5    3.4
```

```
1.6   3.21  3.466  3.743   3.51   3.41
```

6.1.6 Eine kleine Beispieldatei im Hex-Editor:

```
Adresse:  hexadezimale Werte:                      ASCII:
00000000: 5565 6275 6e67 2043 2f43 2b2b 204c 4500  Uebung C/C++ LE.
00000010: 0600 7363 686f 656e 6520 5a61 686c 0078  ..schoene Zahl.x
00000020: 5634 1208 0072 756e 6465 5a61 686c 0004  V4...rundeZahl..
00000030: 0810 2040 80f0 3f08 0050 4900 1c77 4254  .. @..?..PI..wBT
00000040: fb21 0940 0700 7069 00db 0f49 40         .!.@..pi...I@
```

(Anmerkung zur obigen Darstellung: Im linken Block stehen die Hexadezimaladressen innerhalb der Datei, im mittleren Block die Hexadzimalwerte von 16 Byte je Zeile paarweise nebeneinander, und im rechten Block die 16 Zeichen je Zeile zu den Hexadzimalwerten des mittleren Blocks, falls druckbar; sonst ein Punkt. Man kann eine solche Darstellung mit einem sogenannten Hexeditor erreichen, oder indem man die Binärdatei in dem EMACS lädt und dort mit ver+M-x hexl-mode+ die Darstellung wechselt.)

Die ersten 16 Zeichen (ab Adresse `00000000`; erste Zeile) sind die Dateikennung (hexadezimal 0x55, 0x65, 0x62, ..., 0x45, 0x00; beziehungsweise die Zeichenfolge `"Uebung C/C++ LE"`) mit einer abschließenden 0, die im rechten Block („`ASCII`") als Punkt dargestellt ist. An dem Zeichen `'L'` erkennt man, daß die Datei auf einem *little endian System* erstellt wurde (siehe oben), was für die Reihenfolge der folgenden Bytes wichtig ist.

Ab der Adresse `00000010` (hexadezimal, also ab dezimal 16) stehen die beiden Byte 0x06 und 0x00. Dies ist der Typ des ersten Einzelwerts in der Datei. Weil die Datei *little endian* geschrieben ist, ist das erste Byte (0x06) der niederwertige Teil der zwei Byte langen Kennung, das zweite Byte (0x00) der höherwertige Teil. Die Typkennung ist demzufolge `0x0006`; also hat der fragliche Wert den Typ `SLONG`.

Nach diesen beiden Byte folgt ein nullterminierter String, der den Namen des Werts angibt (Adressen `00000010` bis `0000001E`); der Name ist also `schoene Zahl`.

Die dann folgenden 4 Byte an den Adressen `0000001F` bis `00000022` (`0x78`, `0x56`, `0x34` und `0x12`) sind der Wert. Wegen little endian ist das erste Byte das niederwertigste, das letzte Byte ist das höchstwertige. Der Wert ist also `0x12345678`.

Ab der Adresse `00000023` folgt der zweite Wert. Der Typ setzt sich aus `0x08` und `0x00` zusammen; ist also `0x0008` (`DOUBLE`). Der folgende nullterminierte String (`rundeZahl`) ist der Name des zweiten Werts; die dann folgenden 8 Byte (`0x04`, `0x08`, `0x10`, `0x20`, `0x40`, `0x80`, `0xf0`, `0x3f`) sind das Bitmuster der Zahl; wegen little endian ist dieses zur Zahl `0x3ff0804020100804` zusammen zu bauen (und entspricht der binären Gleitkommazahl `1.0000100000000100000000100000000100000000100000000100*2^0`).

Analog folgen noch eine `DOUBLE` mit dem Namen `PI` (Typ `0x0008`, Bitmuster `0x400921fb5442771c`) und eine `FLOAT` (Typ `0x0007`, Bitmuster `0x40490fdb`).

Zum Vergleich die selbe Datei, aber auf einem *big endian System* erzeugt:

```
Adresse:  hexadezimale Werte:                      ASCII:
00000000: 5565 6275 6e67 2043 2f43 2b2b 2042 4500  Uebung C/C++ BE.
00000010: 0006 7363 686f 656e 6520 5a61 686c 0012  ..schoene Zahl..
00000020: 3456 7800 0872 756e 6465 5a61 686c 003f  4Vx..rundeZahl.?
00000030: f080 4020 1008 0400 0850 4900 4009 21fb  ..@ .....PI.@.!.
00000040: 5442 771c 0007 7069 0040 490f db         TBw...pi.@I..
```

Zur Kennzeichnung, daß die folgenden Daten *big endian* zu verstehen sind, wurde vom erzeugenden Programm hier als Kennung der String `"Uebung C/C++ BE"` geschrieben.

Dadurch bleiben die in der Datei enthaltenen Strings unverändert; allerdings ist bei allen `SWORD`, `UWORD`, `SLONG`, `ULONG`, `FLOAT` und `DOUBLE` die Reihenfolge der Byte vertauscht. In diesem Fall müssen also sowohl die Typkennungen als auch alle Werte (außer `STRING`) entsprechend interpretiert werden.

Das zu erstellende Programm muß nun mit beiden Dateiarten zurecht kommen. Je nachdem, ob in der Kennung am Anfang der Datei ein `'B'` oder ein `'L'` steht, ist der Rest der Datei folgerichtig zu behandeln.

6.1.7 Sonstige Anforderungen:

- Die zu lesenden Dateien (möglicherweise mehrere) werden als Parameter an das Programm übergeben.
- Das Programm soll zur Laufzeit feststellen, ob das verwendete Zahlenformat des aktuellen Systems IEEE754 entspricht (für die FLOAT- und DOUBLE-Werte), und mit einer Fehlermeldung abbrechen, falls dies nicht der Fall ist.
- Das Programm muß dafür sorgen, daß seine ganzzahligen Datentypen hinreichend für die gelesenen Werte sind. Zur Vereinfachung ist es zulässig, für die obigen Datentypen (SBYTE, UBYTE, ..., ULONG) mit `typedef` die Typen `char`, `short` und `int` (jeweils `signed` und `unsigned`) auf eigene Namen abzubilden, und zur Laufzeit zu prüfen, ob die so definierten Typen 1, 2 beziehungsweise 4 Byte groß sind (andernfalls das Programm mit einer Fehlermeldung terminieren sollte). Bei Bedarf könnte dann im Fehlerfall durch eine Änderung der `typedef`-Anweisungen das Programm korrigiert werden.

  Besser wäre es natürlich, sich nicht auf die Größen der eingebauten Typen verlassen zu müssen...
- Das Programm soll zur Laufzeit feststellen, ob das verwendete Zahlenformat des aktuellen Systems **big endian** oder **little endian** ist, und im weiteren dann die Werte entsprechend lesen.
- Das Lesen soll möglichst effektiv stattfinden.
- Die gelesenen Daten sollen zur Standardausgabe geschrieben werden. Beispielausgabe:

  ```
  Lesen der Datei "uebung_bin_001_b.dat"
  (Format big endian)
  ```

```
Variable (0x0007, FLOAT)
pi=3.141592

Variable (0x0007, DOUBLE)
klein=-1.678938e-12

Variable (0x0000, STRING)
ichbin="klaus"

Tabelle (0x1047, 3*UWORD/FLOAT)
"einetabelle"
10  3.141593
20  7.234455
25 -1.488357e28

Tabelle (0x1000, , 4*STRING/STRING)
"eineanderetabelle"
"Strauß"  "tot"
"Kohl"    "weg"
"Stoiber" "auch weg"
"Merkel"  "kommt noch"

zweidimensionale Tabelle (0x2787, x=4*FLOAT, y=5*DOUBLE, Werte=FLOAT)
"nocheinetabelle"
x/y   1.0   2.0    2.5     2.6    2.8
1.0   3.0   3.256  3.533   3.3    3.2
1.2   3.1   3.356  3.633   3.4    3.3
1.4   3.2   3.456  3.733   3.5    3.4
1.6   3.21  3.466  3.743   3.51   3.41


Lesen der Datei "uebung_bin_002_l.dat"
...
```

Als Anregung und zum Spicken das Programm, mit dem die Beispieldateien erzeugt wurden:

```
/* Time-stamp: "30.10.02 17:04 ueb8gen.c klaus@wachtler.de"
 *
 **********************************************************************
 *
 * Erzeugt Beispieldateien für die C-Übung "Lesen von Binärdaten auf
 * verschiedenen Systemen"
 *
 **********************************************************************
 *
 * Vom Programm wird geprüft, ob die vereinbarten Typen SBYTE...DOUBLE
 * jeweils wie gefordert 1, 2, 4, oder 8 Byte groß sind.
 * Ist das nicht der Flal, dann wird mit einer Fehlermeldung
 * abgebrochen.
```

```
 * In diesem Fall muß man die typedef-Anweisungen entsprend anpassen
 * und neu übersetzen.
 *
 * Weiterhin wird grob geprüft, ob das aktuelle System float und
 * double entsprechend IEEE754 verwendet.
 * Wenn nicht, wird ebenfalls abgebrochen.
 *
 * Ansonsten sollte dieses Programm auf jedem ANSI-System laufen;
 * je nach System (big oder little endian) unterscheiden sich
 * aber die geschriebenen Dateien.
 *
 * Siehe auch http://www.wachtler.de/fhai/fhai.html (8. C-Aufgabe).
 *
 ********************************************************************
 *
 * Unter Linux getestet mit gcc 2.95-3 (gcc -Wall ueb8gen.c -lm)
 * sowie unter Windows 2000 mit VC++ 6.0 (cl ueb8gen.c).
 *
 ********************************************************************
 */

#include <stdio.h>
#include <stddef.h>
#include <stdlib.h>
#include <errno.h>
#include <math.h>

/* Umsetzung der internen Datentypen auf die gewünschten
 * (muß ggf. für andere Systeme angepaßt werden):
 */
typedef signed char    SBYTE;   /* 1 Byte, mit Vorzeichen       */
typedef unsigned char  UBYTE;   /* 1 Byte, ohne Vorzeichen      */
typedef signed short   SWORD;   /* 2 Byte, mit Vorzeichen       */
typedef unsigned short UWORD;   /* 2 Byte, ohne Vorzeichen      */
typedef signed int     SLONG;   /* 4 Byte, mit Vorzeichen       */
typedef unsigned int   ULONG;   /* 4 Byte, ohne Vorzeichen      */
typedef float          FLOAT;   /* 4 Byte, single gem. IEEE754  */
typedef double         DOUBLE;  /* 8 Byte, double gem. IEEE754  */

/* Hier merken wir uns, ob das aktuelle System little endian ist
 * (1) oder big endian (0):
 */
static int isLittleEndian = -1;


/* öffnet eine Datei zum Schreiben:
 * Der Header "Uebung C/C++ BE\0" bzw. "Uebung C/C++ LE\0" wird gleich
 * geschrieben; je nachdem ob der aktuelle Rechner ein big endian-
 * oder little endian-System ist.
 * Zurückgegeben wird ein FILE-Zeiger, mit dme dann weiter geschrieben
 * werden kann, im Fehlerfall NULL.
 */

FILE *fopen_ueb8_w( const char *fname )
```

```
{
  FILE *f = NULL;

  if( (f = fopen( fname, "wb" )) ) /* Datei zum Schreiben öffnen */
  { /* Öffnen hat geklappt */


    if( fprintf( f,
                 "Uebung C/C++ %cE%c",
                 isLittleEndian ? 'L' : 'B',
                 0
                 )
        !=16
        )
    {
      /* Schreibfehler! */
      fprintf( stderr, "Schreibfehler in Datei!\n" );
      fclose( f ); f = NULL;
      remove( fname );
      errno = 0;
    }
  } /* Öffnen hat geklappt */

  return f;

} /* fopen_ueb8_w() */


/* macht ein paar Initialisierungen (Test, ob die Voraussetzungen
 * erfüllt sind).
 * Rückgabe: 0, wenn Fehler; sonst 1
 */
int init()
{
  /* laufen wir auf einem little endian System (Intel), oder auf
   * einem big endian System (fast alle anderen)?
   * Auf einem little endian System liegt bei einer int mit dem
   * Wert 0x87654321 das Byte mit Wert 0x21 ganz unten; bei einem
   * big endian system dagegen oben.
   */
  union
  { ULONG            i;
    unsigned char    c[sizeof(ULONG)];
  } endian_dummy;
  endian_dummy.i = 0x87654321;
  isLittleEndian = ( endian_dummy.c[0]==0x21 );

#if 0 /* Testausgabe */
  {
    int i;
    printf( "endian_dummy.c[]=\n" );
    for( i=0; i<8; i++ )
    {
      printf( "c[%d] = 0x%02x\n", i, endian_dummy.c[i] );
```

```
    }
  }
  printf( "%s endian\n", isLittleEndian ? "little" : "big" );
#endif

  /* Größen der Datentypen prüfen: */
  if( sizeof(SBYTE)!=1 || sizeof(UBYTE)!=1 )
  {
    fprintf( stderr, "Größe von UBYTE und SBYTE ist nicht 1 Byte!\n" );
    return 0;  /* Fehler! */
  }
  if( sizeof(SWORD)!=2 || sizeof(UWORD)!=2 )
  {
    fprintf( stderr, "Größe von UWORD und SWORD ist nicht 2 Byte!\n" );
    return 0;  /* Fehler! */
  }
  if( sizeof(SLONG)!=4 || sizeof(ULONG)!=4 )
  {
    fprintf( stderr, "Größe von ULONG und SLONG ist nicht 4 Byte!\n" );
    return 0;  /* Fehler! */
  }
  if( sizeof(FLOAT)!=4 )
  {
    fprintf( stderr, "Größe von FLOAT ist nicht 4 Byte!\n" );
    return 0;  /* Fehler! */
  }
  if( sizeof(DOUBLE)!=8 )
  {
    fprintf( stderr, "Größe von DOUBLE ist nicht 8 Byte!\n" );
    return 0;  /* Fehler! */
  }



  /* Mal antesten, ob der aktuelle Rechner double precision gemäß
   * IEEE754 hat (dann glauben wir unbesehen, daß float auch paßt):
   */
  {
    union { double d; unsigned char c[8]; } Versuchskaninchen;
    /* Prüfen, ob die Bitmuster IEEE 754 entsprechen.
     * In einer union liegen eine Gleitkommazahl (8 Byte)
     * und ein char[]-Feld parallel.
     * Wenn man jetzt bestimmte Werte in die Gleitkommazahl schreibt,
     * kann man im char[]-Feld die Werte der einzelnen Bytes angucken.
     *
     * Aufbau einer double nach IEEE ([3], [4]):
     *
     * Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
     * Bit  6666555555555544444444443333333333222222222211111111110000000000
     *      3210987654321098765432109876543210987654321098765432109876543210
     *      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
     *
     * Bit  63 (s) ist das Vorzeichen (0 ist '+', 1 ist '-')
     * Die 11 Bits 52-62 (e) stellen den Exponenten dar, verschoben um
```

```
* den Wert 1023 ins Positive.
* Die 52 Bits 0-51 (m) sind die Mantisse.
* Bei normalisierten Zahlen (Wert für den Exponenten von -1022 bis 1023,
* verschoben in den Bereich 1-2046 für e) wird vor der Binärziffernfolge
* von m eine "1." gedacht, und nicht mit abgespeichert.
*
* Beispiel 1:
* Die Dezimalzahl 1.0 ist binär ebenfalls 1.0, also 1.0*2^0.
* Die "1." wird nicht gespeichert, die gesamte Mantisse m ist 0.
* Der Exponent ist 0, und wird um 1023(dez)=1111111111(binär)
* verschoben.
* Das ergibt:
* Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
* Bit  6666555555555544444444443333333333222222222211111111111
*      3210987654321098765432109876543210987654321098765432109876543210
*      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
* bin: 0011111111110000000000000000000000000000000000000000000000000000
* hex:    3   f   f   0   0   0   0   0   0   0   0   0   0   0   0   0
*
* Beispiel 2:
* Die Dezimalzahl 1.03125 (1+2^-5) ist binär 1.00001, also 1.00001*2^0.
* Die "1." wird nicht gespeichert, die gesamte Mantisse ist 0000100...
* Der Exponent ist 0, und wird um 1023(dez)=1111111111(binär)
* verschoben.
* Das ergibt:
* Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
* Bit  6666555555555544444444443333333333222222222211111111111
*      3210987654321098765432109876543210987654321098765432109876543210
*      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
* bin: 0011111111110000100000000000000000000000000000000000000000000000
* hex:    3   f   f   0   8   0   0   0   0   0   0   0   0   0   0   0
*
* Beispiel 3:
* Die Dezimalzahl
* 1+(2^(-5))+(2^(-14))+(2^(-23))+(2^(-32))+(2^(-41))+(2^(-50))
* ist binär
* 1.0000100000000100000000100000000100000000100000000100,
* also 1.0000100000000100000000100000000100000000100000000100*2^0.
* Die "1." wird nicht gespeichert, die gesamte Mantisse ist 0000100...
* Der Exponent ist 0, und wird um 1023(dez)=1111111111(binär)
* verschoben.
* Das ergibt:
* Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
* Bit  6666555555555544444444443333333333222222222211111111111
*      3210987654321098765432109876543210987654321098765432109876543210
*      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
* bin: 0011111111110000100000000100000000100000000100000000100000000100
* hex:    3   f   f   0   8   0   4   0   2   0   1   0   0   8   0   4
*
* Die Zahl des letzten Beispiels wird als Test verwendet, ob der
* aktuelle Rechner IEEE-konform ist.
* Da hier nur mit Zweierpotenzen gearbeitet wird, sollten
* (IEEE-Konformität vorausgesetzt) keine Rundungfehler auftreten.
*
```

```
 *
 * Wenn man an Versuchskaninchen.d den Wert
 * 1+(2^(-5))+(2^(-14))+(2^(-23))+(2^(-32))+(2^(-41))+(2^(-50))
 * zuweist, dann stehen auf einer little endian CPU danach in
 * Versuchskaninchen.c[0..7] die Werte 0x04, 0x08, 0x10, 0x20,
 * 0x40, 0x80, 0xf0 und 0x3f;
 * auf einer big endian dagegen die Werte 0x3f,
 * 0xf0, 0x80, 0x40, 0x20, 0x10, 0x08, und 0x04.
 */

Versuchskaninchen.d = ( 1.0
                        + pow( 2.0, -5 )
                        + pow( 2.0, -14 )
                        + pow( 2.0, -23 )
                        + pow( 2.0, -32 )
                        + pow( 2.0, -41 )
                        + pow( 2.0, -50 )
                        );

if( !( isLittleEndian
       ? ( Versuchskaninchen.c[0]==0x04
           &&
           Versuchskaninchen.c[1]==0x08
           &&
           Versuchskaninchen.c[2]==0x10
           &&
           Versuchskaninchen.c[3]==0x20
           &&
           Versuchskaninchen.c[4]==0x40
           &&
           Versuchskaninchen.c[5]==0x80
           &&
           Versuchskaninchen.c[6]==0xf0
           &&
           Versuchskaninchen.c[7]==0x3f
           )
       : ( Versuchskaninchen.c[0]==0x3f
           &&
           Versuchskaninchen.c[1]==0xf0
           &&
           Versuchskaninchen.c[2]==0x80
           &&
           Versuchskaninchen.c[3]==0x40
           &&
           Versuchskaninchen.c[4]==0x20
           &&
           Versuchskaninchen.c[5]==0x10
           &&
           Versuchskaninchen.c[6]==0x08
           &&
           Versuchskaninchen.c[7]==0x04
           )
       )
    )
```

```
    {
      /* nein, nicht IEEE754 */
      fprintf( stderr, "double entspricht nicht IEEE754!\n" );
      return 0;  /* Fehler! */
    }

  } /* Kontrolle IEEE 754 */


  return 1;    /* alles ok! */

} /* init() */


/* schreibt in die angegebene Datei einen STRING;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeSTRING( FILE *f, const char *name, const char * wert )
{
  UWORD   kennung = (UWORD)0x0000;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fputs( wert, f )==EOF
      ||
      fputc( 0, f )==EOF
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeSTRING()\n" );
    return 0;
  }


  return 1;  /* alles ok */

} /* schreibeSTRING() */


/* schreibt in die angegebene Datei eine SBYTE;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeSBYTE( FILE *f, const char *name, SBYTE wert )
{
  UWORD   kennung = (UWORD)0x0001;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
```

```
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeSBYTE()\n" );
    return 0;
  }

  return 1;  /* alles ok */

} /* schreibeSBYTE() */


/* schreibt in die angegebene Datei eine UBYTE;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeUBYTE( FILE *f, const char *name, UBYTE wert )
{
  UWORD   kennung = (UWORD)0x0002;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeUBYTE()\n" );
    return 0;
  }

  return 1;  /* alles ok */

} /* schreibeUBYTE() */



/* schreibt in die angegebene Datei eine SWORD;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeSWORD( FILE *f, const char *name, SWORD wert )
{
  UWORD   kennung = (UWORD)0x0003;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
```

```
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeSWORD()\n" );
    return 0;
  }

  return 1;  /* alles ok */

} /* schreibeSWORD() */


/* schreibt in die angegebene Datei eine UWORD;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeUWORD( FILE *f, const char *name, UWORD wert )
{
  UWORD   kennung = (UWORD)0x0004;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeUWORD()\n" );
    return 0;
  }

  return 1;  /* alles ok */

} /* schreibeUWORD() */



/* schreibt in die angegebene Datei eine SLONG;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeSLONG( FILE *f, const char *name, SLONG wert )
{
  UWORD   kennung = (UWORD)0x0005;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeSLONG()\n" );
    return 0;
```

```
  }

  return 1;  /* alles ok */

} /* schreibeSLONG() */


/* schreibt in die angegebene Datei eine ULONG;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeULONG( FILE *f, const char *name, ULONG wert )
{
  UWORD   kennung = (UWORD)0x0006;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeULONG()\n" );
    return 0;
  }


  return 1;  /* alles ok */

} /* schreibeULONG() */



/* schreibt in die angegebene Datei eine FLOAT;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeFLOAT( FILE *f, const char *name, double dwert )
{
  FLOAT   wert = dwert;
  UWORD   kennung = (UWORD)0x0007;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeFLOAT()\n" );
    return 0;
  }
```

```
  return 1;  /* alles ok */

} /* schreibeFLOAT() */


/* schreibt in die angegebene Datei eine DOUBLE;
 * Rückgabe 0, falls Fehler; sonst 1
 */
int schreibeDOUBLE( FILE *f, const char *name, DOUBLE wert )
{
  UWORD   kennung = (UWORD)0x0008;
  if( fwrite( &kennung, sizeof(UWORD), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      ||
      fwrite( &wert, sizeof(wert), 1, f )!=1
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in schreibeDOUBLE()\n" );
    return 0;
  }

  return 1;  /* alles ok */

} /* schreibeDOUBLE() */


int main( int nargs, char **args )
{
  if( !init() )
  {
    /* Abbruch, weil irgendwelche Voraussetzungen nicht erfüllt sind:
     */
    fprintf( stderr, "Abbruch!\n" );
    exit( 1 );
  }

  /* erste Datei mit einer ULONG, zwei DOUBLE und einer FLOAT schreiben: */
  {
    FILE *f = NULL;
    char  dateiname[1024];

    sprintf( dateiname,
             "uebung_bin_simple_%c.dat",
             isLittleEndian ? 'l' : 'b'
             );
    if( !(f=fopen_ueb8_w( dateiname )) )
    {
      fprintf( stderr, "Ich kann die Datei nicht öffnen!\n" );
      exit( 1 );
```

```
  }
  schreibeULONG( f, "schoene Zahl", 0x12345678 );
  schreibeDOUBLE( f,
                  "rundeZahl",
                  1.0
                  + pow( 2.0, -5 )
                  + pow( 2.0, -14 )
                  + pow( 2.0, -23 )
                  + pow( 2.0, -32 )
                  + pow( 2.0, -41 )
                  + pow( 2.0, -50 )
                  );
  schreibeDOUBLE( f,
                  "PI",
                  3.14159265354
                  );
  schreibeFLOAT( f,
                 "pi",
                 3.14159265354
                 );
  fclose( f ); f = NULL;
}

/* Datei für die Beispielausgabe der Aufgabenstellung: */
{
  FILE *f = NULL;
  char  dateiname[1024];

  sprintf( dateiname,
           "uebung_bin_001_%c.dat",
           isLittleEndian ? 'l' : 'b'
           );
  if( !(f=fopen_ueb8_w( dateiname )) )
  {
    fprintf( stderr, "Ich kann die Datei nicht öffnen!\n" );
    exit( 1 );
  }
  schreibeFLOAT( f, "pi", 3.141592 );
  schreibeDOUBLE( f, "klein", 1.678938e-12 );
  schreibeSTRING( f, "ichbin", "klaus" );

  {
    /* Tabelle UWORD,FLOAT: */
    UWORD         kennung = (UWORD)0x1047;
    const ULONG   anzahl  = 3;
    const char   *name    = "eineTabelle";
    UWORD         xwerte[3] = { 10, 20, 30 };
    FLOAT         ywerte[3] = { 3.141593, 7.234455, -1.488357e28 };
    int           i;

    if( fwrite( &kennung, sizeof(kennung), 1, f )!=1
        ||
        fwrite( &anzahl, sizeof(anzahl), 1, f )!=1
        ||
```

```
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in main()\n" );
    exit( 1 );
  }

  for( i=0; i<anzahl; i++ )
  {
    if( fwrite( &xwerte[i], sizeof(xwerte[i]), 1, f )!=1
        ||
        fwrite( &ywerte[i], sizeof(ywerte[i]), 1, f )!=1
        )
    {
      /* Fehler beim Schreiben! */
      fprintf( stderr, "Schreibfehler in main()\n" );
      exit( 1 );
    }
  }
}


{
  /* Tabelle STRING,STRING: */
  UWORD         kennung = (UWORD)0x1000;
  const ULONG   anzahl  = 4;
  const char   *name    = "eineandereTabelle";
  const char   *xwerte[4] = { "FJS", "Kohlibakterie", "Edi", "Angie"  };
  const char   *ywerte[4] = { "tot", "weg", "auch weg", "in Arbeit" };
  int           i;

  if( fwrite( &kennung, sizeof(kennung), 1, f )!=1
      ||
      fwrite( &anzahl, sizeof(anzahl), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in main()\n" );
    exit( 1 );
  }

  for( i=0; i<anzahl; i++ )
  {
    if( fputs( xwerte[i], f )==EOF
        ||
        fputc( 0, f )==EOF
        ||
```

```
        fputs( ywerte[i], f )==EOF
        ||
        fputc( 0, f )==EOF
        )
    {
      /* Fehler beim Schreiben! */
      fprintf( stderr, "Schreibfehler in main()\n" );
      exit( 1 );
    }
  }
}

{
  /* Tabelle FLOATxDOUBLE->FLOAT: */
  UWORD         kennung = (UWORD)0x2787;
  const ULONG   anzahlx = 4;
  const ULONG   anzahly = 5;
  const char   *name    = "nochEineTabelle";
  FLOAT         xwerte[4] = { 1.0, 1.2, 1.4, 1.6 };
  DOUBLE        ywerte[5] = { 1.0, 2.0, 2.5, 2.6, 2.8 };
  FLOAT         zwerte[5][4] =
    {
      { 3.0,   3.1,   3.2,   3.21  },
      { 3.256, 3.356, 3.456, 3.466 },
      { 3.533, 3.633, 3.733, 3.743 },
      { 3.3,   3.4,   3.5,   3.51  },
      { 2.8,   3.2,   3.3,   3.4   },
    };
  int           i, j;

  if( fwrite( &kennung, sizeof(kennung), 1, f )!=1
      ||
      fwrite( &anzahlx, sizeof(anzahlx), 1, f )!=1
      ||
      fwrite( &anzahly, sizeof(anzahly), 1, f )!=1
      ||
      fputs( name, f )==EOF
      ||
      fputc( 0, f )==EOF
      )
  {
    /* Fehler beim Schreiben! */
    fprintf( stderr, "Schreibfehler in main()\n" );
    exit( 1 );
  }

  /* x-Werte: */
  for( i=0; i<anzahlx; i++ )
  {
    if( fwrite( &xwerte[i], sizeof(xwerte[i]), 1, f )!=1
        )
    {
      /* Fehler beim Schreiben! */
      fprintf( stderr, "Schreibfehler in main()\n" );
```

```
          exit( 1 );
        }
      }

      /* y-Werte: */
      for( j=0; j<anzahly; j++ )
      {
        if( fwrite( &ywerte[j], sizeof(ywerte[j]), 1, f )!=1
            )
        {
          /* Fehler beim Schreiben! */
          fprintf( stderr, "Schreibfehler in main()\n" );
          exit( 1 );
        }
      }

      /* z-Werte: */
      for( j=0; j<anzahly; j++ )
      {
        for( i=0; i<anzahlx; i++ )
        {
          if( fwrite( &zwerte[i][j], sizeof(zwerte[i][j]), 1, f )!=1
              )
          {
            /* Fehler beim Schreiben! */
            fprintf( stderr, "Schreibfehler in main()\n" );
            exit( 1 );
          }
        }
      }

    }

    schreibeSLONG( f, "der ganze Rest", 42 );
    schreibeSLONG( f, "Rest andersum", -42 );


    fclose( f ); f = NULL;
  }


  return 0;

} /* main( int nargs, char **args ) */
```

6.2 Lösung: Lesen von Binärdaten auf verschiedenen Systemen

```
/////////////////////////////////////////////////////////////////
//
// Time-stamp: "20.11.02 16:34 lesebinaer.c klaus@wachtler.de"
//
/////////////////////////////////////////////////////////////////
//
// Musterlösung für die C-Übung:
// 8. C-Übung: Lesen von Binärdaten auf verschiedenen Systemen
//
// Es ist ein Programm zu erstellen, welches Binärdateien liest,
// die unter Sun (big endian) oder Intel (little endian) erzeugt
// wurden, und den Inhalt der Dateien in Text umwandelt.
//
// Zum Testen werden einige solche Dateien zur Verfügung
// gestellt.
// ...
//
// Siehe readme.txt!
//
/////////////////////////////////////////////////////////////////
//
// Getestet unter Linux (gcc version 3.0.1, version 2.95.3):
//   gcc -Wall lesebinaer.c -lm && a.out *.dat
// sowie unter Windows 2000 (VC++ 6.0 SP1):
//   cl lesebinaer.c & .\lesebinaer
//
/////////////////////////////////////////////////////////////////
//
// 20.11.2002 Klaus Wachtler  neu erzeugt (passagenweise aus
//                            ueb8gen.c übernommen)
//
/////////////////////////////////////////////////////////////////

#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <ctype.h>

// gibt eine kurze Hilfe aus:
void usage( void )
{
  puts( "Dieses Programm ist die Musterlösung der 8. C-Übung." );
  puts( "Es liest alle als Argument angegebenen Binärdateien" );
  puts( "und schreibt Ihren Inhalt zur Standardausgabe." );
  puts( "(C) 2002 Bin-Laden: Bits zum Mitnehmen" );
}


// logischen Datentyp machen wir uns selbst:
typedef enum
```

```
{
  false, true
} bool;


//Kleinst- oder Größtwert liefern:
#define MIN(a,b) ((a)<(b)?(a):(b))
#define MAX(a,b) ((a)>(b)?(a):(b))


/* Umsetzung der internen Datentypen auf die gewünschten
 * (muß ggf. für andere Systeme angepaßt werden):
 */
typedef signed char    SBYTE;   /* 1 Byte, mit Vorzeichen       */
typedef unsigned char  UBYTE;   /* 1 Byte, ohne Vorzeichen      */
typedef signed short   SWORD;   /* 2 Byte, mit Vorzeichen       */
typedef unsigned short UWORD;   /* 2 Byte, ohne Vorzeichen      */
typedef signed int     SLONG;   /* 4 Byte, mit Vorzeichen       */
typedef unsigned int   ULONG;   /* 4 Byte, ohne Vorzeichen      */
typedef float          FLOAT;   /* 4 Byte, single gem. IEEE754  */
typedef double         DOUBLE;  /* 8 Byte, double gem. IEEE754  */

enum
  {
    STRING_kenn, //  0
    SBYTE_kenn,  //  1
    UBYTE_kenn,  //  2
    SWORD_kenn,  //  3
    UWORD_kenn,  //  4
    SLONG_kenn,  //  5
    ULONG_kenn,  //  6
    FLOAT_kenn,  //  7
    DOUBLE_kenn, //  8
  };

// ein nibble (für den Typ) in einen String übersetzen:
char *typnibble2str[16] =
  {
    "STRING",   //  0
    "SBYTE",    //  1
    "UBYTE",    //  2
    "SWORD",    //  3
    "UWORD",    //  4
    "SLONG",    //  5
    "ULONG",    //  6
    "FLOAT",    //  7
    "DOUBLE",   //  8
    "",         //  9
    "",         // 10
    "",         // 11
    "",         // 12
    "",         // 13
    "",         // 14
    "",         // 15
```

```
  };



// liest die angegebene Anzahl SBYTE aus der Datei f an die
// Stelle p;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_SBYTE( SBYTE *p, size_t n, FILE *f )
{
  return fread( p, sizeof(SBYTE), n, f );
}

// liest die angegebene Anzahl UBYTE aus der Datei f an die
// Stelle p;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_UBYTE( UBYTE *p, size_t n, FILE *f )
{
  return fread( p, sizeof(UBYTE), n, f );
}

// liest nichts, sondern gibt eine Fehlermeldung aus und bricht
// das Programm ab:
int fread_SWORD_err( SWORD *p, size_t n, FILE *f )
{
  fprintf( stderr, "Fehler: fread_SWORD() nicht gesetzt!\n" );
  exit( EXIT_FAILURE );
  return 0;
}

// liest die angegebene Anzahl SWORD aus der Datei f an die
// Stelle p ohne Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_SWORD_direkt( SWORD *p, size_t n, FILE *f )
{
  return fread( p, sizeof(SWORD), n, f );
}

// liest die angegebene Anzahl SWORD aus der Datei f an die
// Stelle p mit Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_SWORD_gedreht( SWORD *p, size_t n, FILE *f )
{
  int i;
  int ngelesen = fread( p, sizeof(SWORD), n, f );

  for( i=0; i<ngelesen; i++ )
  {
    char *byte_p = (char*)&p[i];
    char tmpbyte;
    tmpbyte = byte_p[1];
    byte_p[1] = byte_p[0];
    byte_p[0] = tmpbyte;
  }
  return ngelesen;
```

```
}


// Zeiger auf eine der obigen Funktionen (je nachdem, ob
// konvertiert werden soll, oder nicht):
int (*fread_SWORD)( SWORD *p, size_t n, FILE *f ) = fread_SWORD_err;


int fread_UWORD_err( UWORD *p, size_t n, FILE *f )
{
  fprintf( stderr, "Fehler: fread_UWORD() nicht gesetzt!\n" );
  exit( EXIT_FAILURE );
  return 0;
}

// liest die angegebene Anzahl UWORD aus der Datei f an die
// Stelle p ohne Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_UWORD_direkt( UWORD *p, size_t n, FILE *f )
{
  return fread( p, sizeof(UWORD), n, f );
}

// liest die angegebene Anzahl UWORD aus der Datei f an die
// Stelle p mit Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_UWORD_gedreht( UWORD *p, size_t n, FILE *f )
{
  int i;
  int ngelesen = fread( p, sizeof(UWORD), n, f );

  for( i=0; i<ngelesen; i++ )
  {
    char *byte_p = (char*)&p[i];
    char tmpbyte;
    tmpbyte = byte_p[1];
    byte_p[1] = byte_p[0];
    byte_p[0] = tmpbyte;
  }
  return ngelesen;
}

// Zeiger auf eine der obigen Funktionen (je nachdem, ob
// konvertiert werden soll, oder nicht):
int (*fread_UWORD)( UWORD *p, size_t n, FILE *f ) = fread_UWORD_err;


// liest nichts, sondern gibt eine Fehlermeldung aus und bricht
// das Programm ab:
int fread_SLONG_err( SLONG *p, size_t n, FILE *f )
{
  fprintf( stderr, "Fehler: fread_SLONG() nicht gesetzt!\n" );
  exit( EXIT_FAILURE );
  return 0;
```

```
}

// liest die angegebene Anzahl SLONG aus der Datei f an die
// Stelle p ohne Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_SLONG_direkt( SLONG *p, size_t n, FILE *f )
{
  return fread( p, sizeof(SLONG), n, f );
}

// liest die angegebene Anzahl SLONG aus der Datei f an die
// Stelle p mit Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_SLONG_gedreht( SLONG *p, size_t n, FILE *f )
{
  int i;
  int ngelesen = fread( p, sizeof(SLONG), n, f );

  for( i=0; i<ngelesen; i++ )
  {
    char *byte_p = (char*)&p[i];
    char tmpbyte;

    tmpbyte = byte_p[3];
    byte_p[3] = byte_p[0];
    byte_p[0] = tmpbyte;

    tmpbyte = byte_p[2];
    byte_p[2] = byte_p[1];
    byte_p[1] = tmpbyte;
  }
  return ngelesen;
}

// Zeiger auf eine der obigen Funktionen (je nachdem, ob
// konvertiert werden soll, oder nicht):
int (*fread_SLONG)( SLONG *p, size_t n, FILE *f ) = fread_SLONG_err;


// liest nichts, sondern gibt eine Fehlermeldung aus und bricht
// das Programm ab:
int fread_ULONG_err( ULONG *p, size_t n, FILE *f )
{
  fprintf( stderr, "Fehler: fread_ULONG() nicht gesetzt!\n" );
  exit( EXIT_FAILURE );
  return 0;
}

// liest die angegebene Anzahl ULONG aus der Datei f an die
// Stelle p ohne Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_ULONG_direkt( ULONG *p, size_t n, FILE *f )
{
  return fread( p, sizeof(ULONG), n, f );
```

```
}

// liest die angegebene Anzahl ULONG aus der Datei f an die
// Stelle p mit Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_ULONG_gedreht( ULONG *p, size_t n, FILE *f )
{
  int i;
  int ngelesen = fread( p, sizeof(ULONG), n, f );

  for( i=0; i<ngelesen; i++ )
  {
    char *byte_p = (char*)&p[i];
    char tmpbyte;

    tmpbyte = byte_p[3];
    byte_p[3] = byte_p[0];
    byte_p[0] = tmpbyte;

    tmpbyte = byte_p[2];
    byte_p[2] = byte_p[1];
    byte_p[1] = tmpbyte;
  }
  return ngelesen;
}

// Zeiger auf eine der obigen Funktionen (je nachdem, ob
// konvertiert werden soll, oder nicht):
int (*fread_ULONG)( ULONG *p, size_t n, FILE *f ) = fread_ULONG_err;


// liest nichts, sondern gibt eine Fehlermeldung aus und bricht
// das Programm ab:
int fread_FLOAT_err( FLOAT *p, size_t n, FILE *f )
{
  fprintf( stderr, "Fehler: fread_FLOAT() nicht gesetzt!\n" );
  exit( EXIT_FAILURE );
  return 0;
}

// liest die angegebene Anzahl FLOAT aus der Datei f an die
// Stelle p ohne Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_FLOAT_direkt( FLOAT *p, size_t n, FILE *f )
{
  return fread( p, sizeof(FLOAT), n, f );
}


// liest die angegebene Anzahl FLOAT aus der Datei f an die
// Stelle p mit Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_FLOAT_gedreht( FLOAT *p, size_t n, FILE *f )
{
```

```
  int i;
  int ngelesen = fread( p, sizeof(FLOAT), n, f );

  for( i=0; i<ngelesen; i++ )
  {
    char *byte_p = (char*)&p[i];
    char tmpbyte;

    tmpbyte = byte_p[3];
    byte_p[3] = byte_p[0];
    byte_p[0] = tmpbyte;

    tmpbyte = byte_p[2];
    byte_p[2] = byte_p[1];
    byte_p[1] = tmpbyte;
  }
  return ngelesen;
}

// Zeiger auf eine der obigen Funktionen (je nachdem, ob
// konvertiert werden soll, oder nicht):
int (*fread_FLOAT)( FLOAT *p, size_t n, FILE *f ) = fread_FLOAT_err;

// liest nichts, sondern gibt eine Fehlermeldung aus und bricht
// das Programm ab:
int fread_DOUBLE_err( DOUBLE *p, size_t n, FILE *f )
{
  fprintf( stderr, "Fehler: fread_DOUBLE() nicht gesetzt!\n" );
  exit( EXIT_FAILURE );
  return 0;
}

// liest die angegebene Anzahl DOUBLE aus der Datei f an die
// Stelle p ohne Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_DOUBLE_direkt( DOUBLE *p, size_t n, FILE *f )
{
  return fread( p, sizeof(DOUBLE), n, f );
}

// liest die angegebene Anzahl DOUBLE aus der Datei f an die
// Stelle p mit Konvertierung BE/LE;
// Rückgabe ist die Anzahl der erfolgreich gelesenen Elemente:
int fread_DOUBLE_gedreht( DOUBLE *p, size_t n, FILE *f )
{
  int i;
  int ngelesen = fread( p, sizeof(DOUBLE), n, f );

  for( i=0; i<ngelesen; i++ )
  {
    char *byte_p = (char*)&p[i];
    char tmpbyte;

    tmpbyte = byte_p[7];
```

```
    byte_p[7] = byte_p[0];
    byte_p[0] = tmpbyte;

    tmpbyte = byte_p[6];
    byte_p[6] = byte_p[1];
    byte_p[1] = tmpbyte;

    tmpbyte = byte_p[5];
    byte_p[5] = byte_p[2];
    byte_p[2] = tmpbyte;

    tmpbyte = byte_p[4];
    byte_p[4] = byte_p[3];
    byte_p[3] = tmpbyte;
  }
  return ngelesen;
}

// Zeiger auf eine der obigen Funktionen (je nachdem, ob
// konvertiert werden soll, oder nicht):
int (*fread_DOUBLE)( DOUBLE *p, size_t n, FILE *f ) = fread_DOUBLE_err;


// Hier merken wir uns, ob das aktuelle System little endian ist
// (1) oder big endian (0):
static int isLittleEndian = false;


// macht ein paar Initialisierungen und liefert true, wenn alles
// geklappt hat (und stellt vor allem fest, ob der aktuelle
// Rechner BE oder LE ist, und ob die Datentypen passen).
bool init( void )
{
  /* laufen wir auf einem little endian System (Intel), oder auf
   * einem big endian System (fast alle anderen)?
   * Auf einem little endian System liegt bei einer int mit dem
   * Wert 0x87654321 das Byte mit Wert 0x21 ganz unten; bei einem
   * big endian System dagegen oben.
   */
  union
  { ULONG             i;
    unsigned char     c[sizeof(ULONG)];
  } endian_dummy;
  endian_dummy.i = 0x87654321;
  isLittleEndian = ( endian_dummy.c[0]==0x21 );

  /* Größen der Datentypen prüfen: */
  if( sizeof(SBYTE)!=1 || sizeof(UBYTE)!=1 )
  {
    fprintf( stderr, "Größe von UBYTE und SBYTE ist nicht 1 Byte!\n" );
    return false;  /* Fehler! */
  }
  if( sizeof(SWORD)!=2 || sizeof(UWORD)!=2 )
  {
```

```
  fprintf( stderr, "Größe von UWORD und SWORD ist nicht 2 Byte!\n" );
  return false;  /* Fehler! */
}
if( sizeof(SLONG)!=4 || sizeof(ULONG)!=4 )
{
  fprintf( stderr, "Größe von ULONG und SLONG ist nicht 4 Byte!\n" );
  return false;  /* Fehler! */
}
if( sizeof(FLOAT)!=4 )
{
  fprintf( stderr, "Größe von FLOAT ist nicht 4 Byte!\n" );
  return false;  /* Fehler! */
}
if( sizeof(DOUBLE)!=8 )
{
  fprintf( stderr, "Größe von DOUBLE ist nicht 8 Byte!\n" );
  return false;  /* Fehler! */
}


/* Mal antesten, ob der aktuelle Rechner double precision gemäß
 * IEEE754 hat (dann glauben wir unbesehen, daß float auch paßt):
 */
{
  union { double d; unsigned char c[8]; } Versuchskaninchen;
  /* Prüfen, ob die Bitmuster IEEE 754 entsprechen.
   * In einer union liegen eine Gleitkommazahl (8 Byte)
   * und ein char[]-Feld parallel.
   * Wenn man jetzt bestimmte Werte in die Gleitkommazahl schreibt,
   * kann man im char[]-Feld die Werte der einzelnen Bytes angucken.
   *
   * Aufbau einer double nach IEEE ([3], [4]):
   *
   * Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
   * Bit  666655555555554444444444333333333322222222221111111111
   *      3210987654321098765432109876543210987654321098765432109876543210
   *      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
   *
   * Bit  63 (s) ist das Vorzeichen (0 ist '+', 1 ist '-')
   * Die 11 Bits 52-62 (e) stellen den Exponenten dar, verschoben um
   * den Wert 1023 ins Positive.
   * Die 52 Bits 0-51 (m) sind die Mantisse.
   * Bei normalisierten Zahlen (Wert für den Exponenten von -1022 bis 1023,
   * verschoben in den Bereich 1-2046 für e) wird vor der Binärziffernfolge
   * von m eine "1." gedacht, und nicht mit abgespeichert.
   *
   * Beispiel 1:
   * Die Dezimalzahl 1.0 ist binär ebenfalls 1.0, also 1.0*2^0.
   * Die "1." wird nicht gespeichert, die gesamte Mantisse m ist 0.
   * Der Exponent ist 0, und wird um 1023(dez)=1111111111(binär)
   * verschoben.
   * Das ergibt:
   * Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
   * Bit  666655555555554444444444333333333322222222221111111111
```

```
 *      3210987654321098765432109876543210987654321098765432109876543210
 *      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
 * bin: 0011111111110000000000000000000000000000000000000000000000000000
 * hex:    3   f   f   0   0   0   0   0   0   0   0   0   0   0   0   0
 *
 * Beispiel 2:
 * Die Dezimalzahl 1.03125 (1+2^-5) ist binär 1.00001, also 1.00001*2^0.
 * Die "1." wird nicht gespeichert, die gesamte Mantisse ist 0000100...
 * Der Exponent ist 0, und wird um 1023(dez)=1111111111(binär)
 * verschoben.
 * Das ergibt:
 * Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
 * Bit  6666555555555544444444443333333333222222222211111111111
 *      3210987654321098765432109876543210987654321098765432109876543210
 *      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
 * bin: 0011111111110000100000000000000000000000000000000000000000000000
 * hex:    3   f   f   0   8   0   0   0   0   0   0   0   0   0   0   0
 *
 * Beispiel 3:
 * Die Dezimalzahl
 * 1+(2^(-5))+(2^(-14))+(2^(-23))+(2^(-32))+(2^(-41))+(2^(-50))
 * ist binär
 * 1.00001000000001000000001000000001000000001000000000100,
 * also 1.00001000000001000000001000000001000000001000000000100*2^0.
 * Die "1." wird nicht gespeichert, die gesamte Mantisse ist 0000100...
 * Der Exponent ist 0, und wird um 1023(dez)=1111111111(binär)
 * verschoben.
 * Das ergibt:
 * Byte <---7--><---6--><---5--><---4--><---3--><---2--><---1--><---0-->
 * Bit  6666555555555544444444443333333333222222222211111111111
 *      3210987654321098765432109876543210987654321098765432109876543210
 *      seeeeeeeeeeemmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmmm
 * bin: 0011111111110000100000000100000000100000000100000000100000000100
 * hex:    3   f   f   0   8   0   4   0   2   0   1   0   0   8   0   4
 *
 * Die Zahl des letzten Beispiels wird als Test verwendet, ob der
 * aktuelle Rechner IEEE-konform ist.
 * Da hier nur mit Zweierpotenzen gearbeitet wird, sollten
 * (IEEE-Konformität vorausgesetzt) keine Rundungfehler auftreten.
 *
 *
 * Wenn man an Versuchskaninchen.d den Wert
 * 1+(2^(-5))+(2^(-14))+(2^(-23))+(2^(-32))+(2^(-41))+(2^(-50))
 * zuweist, dann stehen auf einer little endian CPU danach in
 * Versuchskaninchen.c[0..7] die Werte 0x04, 0x08, 0x10, 0x20,
 * 0x40, 0x80, 0xf0 und 0x3f;
 * auf einer big endian dagegen die Werte 0x3f,
 * 0xf0, 0x80, 0x40, 0x20, 0x10, 0x08, und 0x04.
 */


Versuchskaninchen.d = ( 1.0
                        + pow( 2.0, -5 )
                        + pow( 2.0, -14 )
```

```
                              + pow( 2.0, -23 )
                              + pow( 2.0, -32 )
                              + pow( 2.0, -41 )
                              + pow( 2.0, -50 )
                              );

    if( !( isLittleEndian
           ? ( Versuchskaninchen.c[0]==0x04
               &&
               Versuchskaninchen.c[1]==0x08
               &&
               Versuchskaninchen.c[2]==0x10
               &&
               Versuchskaninchen.c[3]==0x20
               &&
               Versuchskaninchen.c[4]==0x40
               &&
               Versuchskaninchen.c[5]==0x80
               &&
               Versuchskaninchen.c[6]==0xf0
               &&
               Versuchskaninchen.c[7]==0x3f
               )
           : ( Versuchskaninchen.c[0]==0x3f
               &&
               Versuchskaninchen.c[1]==0xf0
               &&
               Versuchskaninchen.c[2]==0x80
               &&
               Versuchskaninchen.c[3]==0x40
               &&
               Versuchskaninchen.c[4]==0x20
               &&
               Versuchskaninchen.c[5]==0x10
               &&
               Versuchskaninchen.c[6]==0x08
               &&
               Versuchskaninchen.c[7]==0x04
               )
           )
        )
    {
      /* nein, nicht IEEE754 */
      fprintf( stderr, "double entspricht nicht IEEE754!\n" );
      return false;  /* Fehler! */
    }

  } /* Kontrolle IEEE 754 */

  return true; // alles ok

} // init()
```

```
// liest aus f einen nullterminierten String, und allokiert
// Speicher dafür;
// der Aufrufer ist dafür verantwortlich, den Speicher beizeiten
// mit free() wieder freizugeben.
// Im Fehlerfall wird NULL geliefert.
char *leseString( FILE *f )
{
  char   *allokiert    = NULL;
  size_t  lallokiert   = 0;
  int     c;

  do
  {
    char   *neuallokiert = NULL;
    // Dateiende wie abschließende 0 behandeln:
    if( (c=fgetc( f ))==EOF )
    {
      c = 0;
    }
    // ein Zeichen mehr allokieren:
    if( !(neuallokiert = (char*)realloc( allokiert, ++lallokiert )) )
    {
      free( allokiert );
      return NULL;
    }
    else
    {
      allokiert = neuallokiert;
    }
    allokiert[lallokiert-1] = c;
  }
  while( c );

  return allokiert;

} // leseString()


// hängt an einen allokierten String einen zweiten String an;
// der erste String wird mit realloc() angepaßt.
// Wenn wohin beim Aufruf NULL ist, wird ein neuer String
// erzeugt.
// Rückgabe: neu allokierter Speicher, oder NULL, wenn kein
// Speicher frei.
char *strcatRealloc( char *wohin, const char *woher )
{
  int   vorherleer = ( wohin==NULL );
  char *wohin_neu = realloc( wohin,
                             ( ( vorherleer ? 0 : strlen( wohin ) )
                               + strlen( woher )
                               + 1 )
                             );
  if( wohin_neu )
  {
```

```
    // für einen neuen String strcpy() aufrufen, sonst strcat():
    ( vorherleer ? strcpy : strcat )( wohin_neu, woher );
  }
  else
  {
    free( wohin );
  }
  return wohin_neu;
}

// hängt an einen allokierten String ein Zeichen an;
// der erste String wird mit realloc() angepaßt.
// Wenn wohin beim Aufruf NULL ist, wird ein neuer String
// erzeugt.
// Rückgabe: neu allokierter Speicher, oder NULL, wenn kein
// Speicher frei.
char *strcatCharRealloc( char *wohin, int c )
{
  char puffer[2] = { c, 0 };
  return strcatRealloc( wohin, puffer );
}


// allokiert genug Speicher für eine Kopie von s und kopiert
// s dorthin (falls s NULL ist, wird ein Byte für einen leeren
// String allokiert):
char *strmalloccpy( const char *s )
{
  char *neu = (char *)malloc( ( s ? strlen( s ) : 0 ) + 1 );
  if( !neu )
  {
    fprintf( stderr, "kein Speicher in strmalloccpy()\n" );
    return NULL;
  }
  return strcpy( neu, ( s ? s : "" ) );
}


// liest einen Einzelwert aus f und liefert ihn als passend
// allokierten String (den der Aufrufer mit free() freizugeben
// hat), oder NULL:
char *leseWertAlsString( unsigned typ, FILE *f )
{
  char puffer[100]; // reicht für alles, außer für Strings.


  switch( typ )
  {
    case STRING_kenn:
      {
        // Erst einen String aus Datei lesen.
        // Der Originalstring muß dann zeichenweise umkopiert
        // werden; dabei sollen Sonderzeichen ersetzt werden und
        // vorne und hinten sollen Gänsefüßchen stehen:
```

```
char *wert = leseString( f );
char *wert_ip = wert;
char *kopie = NULL;

if( !( kopie = strcatRealloc( kopie, "\"" ) ) ) // " vorne
{
  free( wert );
  return NULL;
}

while( *wert_ip )
{ // Schleife über alle Zeichen in wert

  switch( *wert_ip )
  {
    case '\'':
      kopie = strcatRealloc( kopie, "\\\'" );
      break;

    case '\"':
      kopie = strcatRealloc( kopie, "\\\"" );
      break;

    case '\t':
      kopie = strcatRealloc( kopie, "\\t" );
      break;

    case '\r':
      kopie = strcatRealloc( kopie, "\\r" );
      break;

    case '\n':
      kopie = strcatRealloc( kopie, "\\n" );
      break;

    case '\a':
      kopie = strcatRealloc( kopie, "\\a" );
      break;

    case '\v':
      kopie = strcatRealloc( kopie, "\\v" );
      break;

    case '\b':
      kopie = strcatRealloc( kopie, "\\b" );
      break;

    case '\f':
      kopie = strcatRealloc( kopie, "\\f" );
      break;

    case '\\':
      kopie = strcatRealloc( kopie, "\\\\" );
      break;
```

```
        case '\?':
          kopie = strcatRealloc( kopie, "\\?" );
          break;

        default :
          if( isprint( *wert_ip ) )
          {
            // druckbar? Einfügen!
            kopie = strcatCharRealloc( kopie, *wert_ip );
          }
          else
          {
            // nicht druckbar? Dann oktal einfügen
            char puffer[5]; // reicht für jedes Zeichen
                            // ("\007")
            sprintf( puffer, "\\%03o", *wert_ip );
            kopie = strcatRealloc( kopie, puffer );
          }
          break;
      }
      if( !kopie )
      {
        free( wert );
        return NULL;
      }

      wert_ip++;

    } // Schleife über alle Zeichen in wert

    if( !( kopie = strcatRealloc( kopie, "\"" ) ) ) // " hinten
    {
      free( wert );
      return NULL;
    }

    free( wert );
    return kopie;

  } // STRING_kenn
  break;

case SBYTE_kenn:
  {
    // Wert aus Datei lesen, in char[] schreiben und
    // allokierte Kopie davon zurückgeben:
    SBYTE   wert;
    if( fread_SBYTE( &wert, 1, f )!=1 )
    {
      fprintf( stderr,
               "Lesefehler in leseWertAlsString( SBYTE_kenn )"
               " bei Position %ld (0x%08lX)\n",
               ftell( f ),
```

```
              ftell( f )
              );
      return NULL;
    }
    sprintf( puffer, "%d", (int)wert );
    return strmalloccpy( puffer );
  }
  break;

case UBYTE_kenn:
  {
    // Wert aus Datei lesen, in char[] schreiben und
    // allokierte Kopie davon zurückgeben:
    UBYTE   wert;
    if( fread_UBYTE( &wert, 1, f )!=1 )
    {
      fprintf( stderr,
              "Lesefehler in leseWertAlsString( UBYTE_kenn )"
              " bei Position %ld (0x%08lX)\n",
              ftell( f ),
              ftell( f )
              );
      return NULL;
    }
    sprintf( puffer, "0x%X", (unsigned)wert );
    return strmalloccpy( puffer );
  }
  break;

case SWORD_kenn:
  {
    // Wert aus Datei lesen, in char[] schreiben und
    // allokierte Kopie davon zurückgeben:
    SWORD   wert;
    if( fread_SWORD( &wert, 1, f )!=1 )
    {
      fprintf( stderr,
              "Lesefehler in leseWertAlsString( SWORD_kenn )"
              " bei Position %ld (0x%08lX)\n",
              ftell( f ),
              ftell( f )
              );
      return NULL;
    }
    sprintf( puffer, "%d", (int)wert );
    return strmalloccpy( puffer );
  }
  break;


case UWORD_kenn:
  {
    // Wert aus Datei lesen, in char[] schreiben und
    // allokierte Kopie davon zurückgeben:
```

```
    UWORD   wert;
    if( fread_UWORD( &wert, 1, f )!=1 )
    {
      fprintf( stderr,
               "Lesefehler in leseWertAlsString( UWORD_kenn )"
               " bei Position %ld (0x%08lX)\n",
               ftell( f ),
               ftell( f )
               );
      return NULL;
    }
    sprintf( puffer, "0x%X", (unsigned)wert );
    return strmalloccpy( puffer );
  }
  break;

case SLONG_kenn:
  {
    // Wert aus Datei lesen, in char[] schreiben und
    // allokierte Kopie davon zurückgeben:
    SLONG   wert;
    if( fread_SLONG( &wert, 1, f )!=1 )
    {
      fprintf( stderr,
               "Lesefehler in leseWertAlsString( SLONG_kenn )"
               " bei Position %ld (0x%08lX)\n",
               ftell( f ),
               ftell( f )
               );
      return NULL;
    }
    sprintf( puffer, "%ld", (long)wert );
    return strmalloccpy( puffer );
  }
  break;

case ULONG_kenn:
  {
    // Wert aus Datei lesen, in char[] schreiben und
    // allokierte Kopie davon zurückgeben:
    ULONG   wert;
    if( fread_ULONG( &wert, 1, f )!=1 )
    {
      fprintf( stderr,
               "Lesefehler in leseWertAlsString( ULONG_kenn )"
               " bei Position %ld (0x%08lX)\n",
               ftell( f ),
               ftell( f )
               );
      return NULL;
    }
    sprintf( puffer, "0x%lX", (unsigned long)wert );
    return strmalloccpy( puffer );
  }
```

```
      break;

    case FLOAT_kenn:
      {
        // Wert aus Datei lesen, in char[] schreiben und
        // allokierte Kopie davon zurückgeben:
        FLOAT   wert;
        if( fread_FLOAT( &wert, 1, f )!=1 )
        {
          fprintf( stderr,
                   "Lesefehler in leseWertAlsString( FLOAT_kenn )"
                   " bei Position %ld (0x%08lX)\n",
                   ftell( f ),
                   ftell( f )
                   );
          return NULL;
        }
        sprintf( puffer, "%g", (double)wert );
        return strmalloccpy( puffer );
      }
      break;

    case DOUBLE_kenn:
      {
        // Wert aus Datei lesen, in char[] schreiben und
        // allokierte Kopie davon zurückgeben:
        DOUBLE   wert;
        if( fread_DOUBLE( &wert, 1, f )!=1 )
        {
          fprintf( stderr,
                   "Lesefehler in leseWertAlsString( DOUBLE_kenn )"
                   " bei Position %ld (0x%08lX)\n",
                   ftell( f ),
                   ftell( f )
                   );
          return NULL;
        }
        sprintf( puffer, "%g", (double)wert );
        return strmalloccpy( puffer );
      }
      break;

    default :
      fprintf( stderr,
               "unbekannter Typ %d in leseWertAlsString()\n",
               typ
               );
      return NULL;
      break;
  }

} // leseWertAlsString()

// liest einen Einzelwert aus der Datei und liefert true, wenn ok:
```

```
bool leseEinzelwert( UWORD kennung, FILE *f )
{
  bool hatsgeklappt = true;
  long alte_Position = ftell( f );
  char *name = leseString( f ); // Name der Variablen
  char *wert = NULL;
  if( !name )
  {
    fprintf( stderr,
             "Fehler beim Lesen des Namens einer Variablen"
             " an der Stelle %ld (0x%08lX)\n",
             alte_Position,
             alte_Position
             );
    return false;
  }

  alte_Position = ftell( f );
  if( ( wert = leseWertAlsString( kennung, f ) ) )
  {
    printf( "Variable (0x%04X, %s)\n",
            kennung,
            typnibble2str[kennung]
            );
    printf( "%s=%s\n", name, wert );
    free( wert );
  }
  else
  {
    fprintf( stderr,
             "Fehler beim Lesen des Namens einer Variablen"
             " an der Stelle %ld (0x%08lX)\n",
             alte_Position,
             alte_Position
             );
    hatsgeklappt = false;
  }

  free( name );


  return hatsgeklappt;

} // leseEinzelwert()


// liest ein eindimensionales Feld aus der Datei und liefert
// true, wenn ok:
bool leseEinDimFeld( UWORD kennung, FILE *f )
{
  // Typ der ersten und zweiten Spalte:
  unsigned xTyp = ( kennung & 0x00F0 )>>4;
  unsigned yTyp = ( kennung & 0x000F );
```

```
ULONG Feldlaenge;
if( fread_ULONG( &Feldlaenge, 1, f )!=1 )
{
  // Feldlänge kann nicht gelesen werden!
  fprintf( stderr,
           "Feldlänge an Position %ld (0x%08lX) nicht lesbar\n",
           ftell( f ),
           ftell( f )
           );
  return false;
}
else
{ // Feldlänge gelesen

  bool hatsgeklappt = true;

  char
    **xStrings = NULL,
    **yStrings = NULL;
  char *name = NULL;
  unsigned long alte_position = ftell( f );

  if( ( xStrings = (char**)calloc( Feldlaenge, sizeof(xStrings[0]) ) )
      &&
      ( yStrings = (char**)calloc( Feldlaenge, sizeof(yStrings[0]) ) )
      &&
      ( name = leseString( f ) )
      )
  { // Allokieren von name, xStrings und yStrings hat geklappt
    size_t i;

    // größte vorkommende Länge:
    size_t xMaxLaenge = 0, yMaxLaenge = 0;

    // Titel ausgeben:
    printf( "Tabelle (0x%04X, %lu*%s/%s)\n\"%s\"\n",
            (unsigned)kennung,
            (unsigned long)Feldlaenge,
            typnibble2str[xTyp],
            typnibble2str[yTyp],
            name
            );

    // Schleife über alle Wertepaare:
    for( i=0; hatsgeklappt && i<Feldlaenge; i++ )
    {
      // den ersten und zweiten Wert eines Paares lesen und zu
      // je einem String machen:
      if( (xStrings[i] = leseWertAlsString( xTyp, f ))
          &&
          (yStrings[i] = leseWertAlsString( yTyp, f ))
          )
      {
        xMaxLaenge = MAX( xMaxLaenge, strlen( xStrings[i] ) );
```

```
      yMaxLaenge = MAX( yMaxLaenge, strlen( yStrings[i] ) );
    }
    else
    {
      hatsgeklappt = false;
    }
  }

  // Jetzt stehen in xStrings[] und yStrings[] die Werte als
  // Strings, und in xMaxLaenge und yMaxLaenge stehen die
  // jeweils größten Stringlängen.
  // Passend ausgeben:
  for( i=0; hatsgeklappt && i<Feldlaenge; i++ )
  {
    printf( "   %-*s %-*s\n",
            (int)xMaxLaenge,
            xStrings[i],
            (int)yMaxLaenge,
            yStrings[i]
            );
  }

  free( name );
  // alle Werte freigeben:
  for( i=0; i<Feldlaenge; i++ )
  {
    free( xStrings[i] );
    free( yStrings[i] );
  }
  free( xStrings );
  free( yStrings );
} // Allokieren von name, xStrings und yStrings hat geklappt
else
{
  // Allokieren von name, xStrings und yStrings gescheitert!
  if( !xStrings )
  {
    fprintf( stderr, "kein Speicher für xStrings frei!\n" );
  }
  else
  {
    if( !yStrings )
    {
      fprintf( stderr, "kein Speicher für yStrings frei!\n" );
    }
    else
    {
      fprintf( stderr,
               "Lesen von name gescheitert an Position %ld (0x%08lX)\n",
               alte_position,
               alte_position
               );
    }
    free( xStrings );
```

```
      }
    }

    return hatsgeklappt;
  } // Feldlänge gelesen

} // leseEinDimFeld()


// liest ein eindimensionales Feld aus der Datei und liefert
// true, wenn ok:
bool leseZweiDimFeld( UWORD kennung, FILE *f )
{
  // Typ der ersten und zweiten Spalte und der Werte:
  unsigned xTyp = ( kennung & 0x0F00 )>>8;
  unsigned yTyp = ( kennung & 0x00F0 )>>4;
  unsigned zTyp = ( kennung & 0x000F );

  ULONG xFeldlaenge;
  ULONG yFeldlaenge;

  printf( "xTyp = 0x%02X\n", xTyp );
  printf( "yTyp = 0x%02X\n", yTyp );
  printf( "zTyp = 0x%02X\n", zTyp );

  if( fread_ULONG( &xFeldlaenge, 1, f )!=1
      ||
      fread_ULONG( &yFeldlaenge, 1, f )!=1
      )
  {
    // Feldlänge kann nicht gelesen werden!
    fprintf( stderr,
             "Feldlänge an Position %ld (0x%08lX) nicht lesbar\n",
             ftell( f ),
             ftell( f )
             );
    return false;
  }
  else
  { // Feldlängen gelesen


    int iZeile, iSpalte;

    bool hatsgeklappt = true;

    char
      **xStrings = NULL,
      **yStrings = NULL,
      **zStrings = NULL;
    char *name = NULL;
    unsigned long alte_position = ftell( f );

    if( ( xStrings = (char**)calloc( xFeldlaenge, sizeof(xStrings[0]) ) )
```

```
    &&
    ( yStrings = (char**)calloc( yFeldlaenge, sizeof(yStrings[0]) ) )
    &&
    ( zStrings = (char**)calloc( xFeldlaenge*yFeldlaenge,
                                 sizeof(yStrings[0])
                                 )
      )
    &&
    ( name = leseString( f ) )
    )
{ // Allokieren von name, xStrings, yStrings und zStrings hat geklappt
  size_t i;

  // größte vorkommende Länge:
  size_t xMaxLaenge = 0, yMaxLaenge = 0;
  // (die Funktionswerte z stehen unter den y-Werten; deshalb
  // wird yMaxLaenge für z mitbenutzt)

  // Titel ausgeben:
  printf( "zweidimensionale Tabelle"
          " (0x%04X, x=%lu*%s, y=%lu*%s, Werte=%s)\n"
          "\"%s\"\n",
          (unsigned)kennung,
          (unsigned long)xFeldlaenge,
          typnibble2str[xTyp],
          (unsigned long)yFeldlaenge,
          typnibble2str[yTyp],
          typnibble2str[zTyp],
          name
          );

  // Schleife über alle x-Werte:
  for( i=0; hatsgeklappt && i<xFeldlaenge; i++ )
  {
    if( (xStrings[i] = leseWertAlsString( xTyp, f )) )
    {
      xMaxLaenge = MAX( xMaxLaenge, strlen( xStrings[i] ) );
    }
    else
    {
      hatsgeklappt = false;
    }
  }

  // Schleife über alle y-Werte:
  for( i=0; hatsgeklappt && i<yFeldlaenge; i++ )
  {
    if( (yStrings[i] = leseWertAlsString( yTyp, f )) )
    {
      yMaxLaenge = MAX( yMaxLaenge, strlen( yStrings[i] ) );
    }
    else
    {
      hatsgeklappt = false;
```

```
  }
}

// Schleife über alle z-Werte:
for( i=0; hatsgeklappt && i<xFeldlaenge*yFeldlaenge; i++ )
{
  if( (zStrings[i] = leseWertAlsString( zTyp, f )) )
  {
    yMaxLaenge = MAX( yMaxLaenge, strlen( zStrings[i] ) );
  }
  else
  {
    hatsgeklappt = false;
  }
}

if( hatsgeklappt )
{ // Ausgabe

  // Jetzt stehen in x/y/zStrings[] die Werte als
  // Strings, und in xMaxLaenge und yMaxLaenge stehen die
  // jeweils größten Stringlängen der Spalten.

  // Kopfzeile ausgeben:
  printf( "   %-*s", xMaxLaenge, "x/y" );
  for( iSpalte=0; iSpalte<yFeldlaenge; iSpalte++ )
  {
    printf( " %-*s", yMaxLaenge, yStrings[iSpalte] );
  }
  printf( "\n" );

  // jetzt den Rest der Tabelle zeilenweise:
  for( iZeile=0; iZeile<xFeldlaenge; iZeile++ )
  {
    printf( "   %-*s", xMaxLaenge, xStrings[i] );
    for( iSpalte=0; iSpalte<yFeldlaenge; iSpalte++ )
    {
      printf( " %-*s",
              yMaxLaenge,
              zStrings[iZeile*yFeldlaenge+iSpalte]
              );
    }
    printf( "\n" );
  }
} // Ausgabe

free( name );
// alle Werte freigeben:
for( i=0; i<xFeldlaenge; i++ )
{
  free( xStrings[i] );
}
for( i=0; i<yFeldlaenge; i++ )
{
```

```
        free( yStrings[i] );
      }
      for( i=0; i<xFeldlaenge*yFeldlaenge; i++ )
      {
        free( zStrings[i] );
      }
      free( xStrings );
      free( yStrings );
      free( zStrings );
    } // Allokieren von name, xStrings und yStrings hat geklappt
    else
    {
      // Allokieren von name, xStrings und yStrings gescheitert!
      if( !xStrings )
      {
        fprintf( stderr, "kein Speicher für xStrings frei!\n" );
      }
      else
      {
        free( xStrings );
        if( !yStrings )
        {
          fprintf( stderr, "kein Speicher für yStrings frei!\n" );
        }
        else
        {
          free( yStrings );
          if( !zStrings )
          {
            fprintf( stderr, "kein Speicher für yStrings frei!\n" );
          }
          else
          {
            free( zStrings );
            fprintf( stderr,
                     "Lesen von name gescheitert an Position %ld (0x%08lX)\n",
                     alte_position,
                     alte_position
                     );
          }
        }
      }
    }


    return hatsgeklappt;
  } // Feldlängen gelesen

} // leseZweiDimFeld()



// liest eine Datei, gibt sie zur Standardausgabe aus, und
// liefert true, wenn alles geklappt hat.
```

```
bool leseDatei( const char* DateiName )
{
  FILE *f = NULL;
  bool  isDateiLE = false;
  bool  hatsgeklappt = true;

  printf( "Lesen der Datei \"%s\"\n", DateiName );

  if( !(f=fopen( DateiName, "rb" )) )
  {
    fprintf( stderr, "Die Datei %s ist nicht lesbar!\n", DateiName );
    return false;
  }

  // die ersten Byte lesen, und sehen, ob der Anfang stimmt:
  {
    char  header[16];

    if( fread( header, 1, sizeof(header), f )!=sizeof(header) )
    {
      fprintf( stderr,
               "Lesefehler in Datei an Position %ld (0x%08lX)\n",
               ftell( f ),
               ftell( f )
               );
      fclose( f ); f = NULL;
      return false;
    }

    if( memcmp( header, "Uebung C/C++ ", 13 )
        ||
        memcmp( header+14, "E", 2 )
        )
    {
      fprintf( stderr, "Dateiheader stimmt nicht!\n" );
      fprintf( stderr, "Das ist keine Datei im Format der C-Übung.\n" );
      fclose( f ); f = NULL;
      return false;
    }

    switch( header[13] )
    {
      case 'L':
        isDateiLE = true;
        break;

      case 'B':
        isDateiLE = false;
        break;

      default :
        fprintf( stderr, "Dateiheader stimmt nicht!\n" );
        fprintf( stderr, "Das ist keine Datei im Format der C-Übung.\n" );
        fclose( f ); f = NULL;
```

```
      return false;
      break;
  }
}

printf( "(Format %s endian)\n",
        isDateiLE ? "little" : "big"
        );

// BE/LE der Datei gleich dem aktuellen System?
if( isDateiLE==isLittleEndian )
{
  // dann mit Funktionen lesen, die nichts umdrehen:
  fread_SWORD = fread_SWORD_direkt;
  fread_UWORD = fread_UWORD_direkt;
  fread_SLONG = fread_SLONG_direkt;
  fread_ULONG = fread_ULONG_direkt;
  fread_FLOAT = fread_FLOAT_direkt;
  fread_DOUBLE = fread_DOUBLE_direkt;
}
else
{
  // Funktionen zum Lesen verwenden, die alle Byte
  // von BE nach LE oder umgekehrt umwandeln:
  fread_SWORD = fread_SWORD_gedreht;
  fread_UWORD = fread_UWORD_gedreht;
  fread_SLONG = fread_SLONG_gedreht;
  fread_ULONG = fread_ULONG_gedreht;
  fread_FLOAT = fread_FLOAT_gedreht;
  fread_DOUBLE = fread_DOUBLE_gedreht;
}


// Ab jetzt kommt eine Folge von Einzelwerten, ein- oder
// zweidimensionalen Feldern in der Datei.
// Alle beginnen mit einer Kennung (UWORD), sowie einem Namen
// (nullterminierter String), je nach Kennung unterscheiden
// sich dann die restlichen Daten.
// Also lesen wir hier die kennung und den Namen, und
// verzweigen dann in speziellere Funktionen, die den Rest
// lesen:
while( true )
{ // Schleife über alle Elemente der Datei

  UWORD kennung;

  if( fread_UWORD( &kennung, 1, f )!=1 )
  {
    break; // ok, Dateiende
  }

  if( ( kennung & 0xFFF0 )==0x0000 )
  {
    // Nur im untersten Nibble steht etwas, also Einzelwert!
```

```
      if( !leseEinzelwert( kennung, f ) )
      {
        fprintf( stderr, "Fehler beim Lesen von Einzelwert\n" );
        hatsgeklappt = false;
        break; // Fehler!
      }
    }
    else if( ( kennung & 0xFF00 )==0x1000 )
    {
      // Oberstes Byte ist 0x10, also eindim. Feld:!
      if( !leseEinDimFeld( kennung, f ) )
      {
        fprintf( stderr, "Fehler beim Lesen von einfachem Feld\n" );
        hatsgeklappt = false;
        break; // Fehler!
      }
    }
    else if( ( kennung & 0xF000 )==0x2000 )
    {
      // Oberstes Byte ist 0x2*, also zweidim. Feld:!
      if( !leseZweiDimFeld( kennung, f ) )
      {
        fprintf( stderr, "Fehler beim Lesen von zweidim. Feld\n" );
        hatsgeklappt = false;
        break; // Fehler!
      }
    }
    else
    {
      fprintf( stderr,
               "unbekannte Kennung 0x%X an Position %ld (0x%08lX)\n",
               kennung,
               ftell( f ) - 2,
               ftell( f ) - 2
               );
      hatsgeklappt = false;
      break;
    }


    printf( "\n" );

  } // Schleife über alle Elemente der Datei

  fclose( f ); f = NULL;
  return hatsgeklappt; // alles ok

} // leseDatei()


int main( int nargs, char **args )
{
  // Initialisierung durchführen:
  if( !init() )
```

```
  {
    return EXIT_FAILURE;
  }

  // Auswerten der Argumente:
  // Jeder Parameter soll als Dateiname verwendet werden
  // (aber [0] übergehen!).
  while( ++args, --nargs )
  {
    if( !strcmp( *args, "-h" )
        ||
        !strcmp( *args, "-?" )
        ||
        !strcmp( *args, "--help" )
        )
    {
      usage();
      return EXIT_SUCCESS;
    }
    else
    {
      leseDatei( *args );
    }
  }

  return EXIT_SUCCESS;
} /* main( int nargs, char **args ) */
```

7 Zuweisung einfacher Objekte

7.1 Aufgabe: Zuweisung einfacher Objekte

Es ist eine einfache Klasse `Person` zu schreiben, die aus mindestens einem Vor- und einem Nachnamen für eine Person besteht.

Es sind mindestens zwei Methoden zu definieren:

- eine Methode `setze( const char *vn, const char *nn )` setzt den Vor- und den Nachnamen eines Objekts
- `zeige()` gibt den aktuellen Inhalt eines Objekts zur Standardausgabe aus.

Die Klasse ist in drei verschiedenen Versionen zu implementieren, wobei sich die interne Darstellung der Namen (Vor- und Nachname) unterscheidet:

1. in der primitivsten Version sollen die Namen als `char`-Felder mit fester Länge vorliegen; damit wird für kurze Namen Platz verschwendet und Namen ab einer gewissen Länge sind nicht möglich
2. die für den Ersteller der Klasse bequemste, aber auch zur Laufzeit uneffektivste Version soll den in C++ vorhandenen Datentyp `string` verwenden, der automatisch für das Allokieren von ausreichend Speicher sorgt
3. die dritte Version soll selbst für das Allokieren sorgen, indem in der Klasse nur ein Zeiger auf die Namen steht, und bei Bedarf mit `new` oder `malloc()` Platz beschafft wird.

   Hierbei ist aber darauf zu achten, daß nicht mehr benötigter Speicher auch sauber freigegeben wird.

Die Klasse soll mit folgendem Testprogramm ausprobiert werden:

```
// Time-stamp: "08.12.02 13:14 test_person.cpp klaus@wachtler.de"
//
// Testprogramm für class Person aus der C++-Übung A

//#include <cstdlib>
//#include <cstddef>
//#include <cstdio>
//#include <cstring>

#include <iostream>

#include <exception>


#include "Person.h"
```

```
using namespace std;

int main( int nargs, char **args )
{
  try
  {
    // zwei Personen vereinbaren und initialisieren:
    cout << "(definieren)" << endl;
    Person a( "Karl", "Moik", 72 ), b( "Otto", "Waalkes", 55 );

    // Beide Personen ausgeben:
    a.zeige(); b.zeige();

    // a zu einer Kopie von b machen:
    cout << "(zuweisen)" << endl;
    a = b;

    // Beide Personen ausgeben:
    a.zeige(); b.zeige();

    // b neu setzen:
    cout << "(b setzen)" << endl;
    b.setze( "Eddi", "Stotter", 88 );

    // Beide Personen ausgeben:
    a.zeige(); b.zeige();
  }
  catch( exception &Fehler )
  {
    cerr << "Ausnahme in main() gefangen:\n"
         << Fehler.what() << endl;
  }
  catch( ... )
  {
    cerr << "unbekannte Ausnahme in main()" << endl;
  }


  return 0;
} // main( int nargs, char **args )
```

(Der Anwender der Klasse wird also nicht damit belästigt, wie die Implementation intern aussieht.)

Themen:

- Zeiger als Klassenelemente
- Allokieren und Freigeben
- Überladen von Operatoren

Spätester Abgabetermin ist Mittwoch, der 08.12.2004. Empfohlen ist eine Bearbeitung bis 01.11.2004.

7.2 Lösung: Zuweisung einfacher Objekte

Die Klasse `Person` ist recht einfach, so daß es sich nicht lohnt, dafür eine eigene CPP-Datei zu schreiben; die Methoden werden gleich in der Klasse und damit in der Headerdatei untergebracht.

Um die Unterschiede zwischen den drei Versionen klarer zu machen, sind in einem Quelltext alle geforderten Varianten untergebracht, und werden mit Präprozessordirektiven aktiviert (durch Aktivieren genau einer der drei Zeilen der Art `#define AKTUELLEVERSION`...):

```
// Time-stamp: "15.12.04 17:37 Person.h klaus@wachtler.de"
//
// Deklarationen (und Definitionen) für die class Person aus der
// C++-Übung A
//
// Musterlösung 08.12.2002 Klaus Wachtler
//
// 15.12.2004 kw   operator== ergänzt für Aufgabe 8

#ifndef __PERSON_H__
#define __PERSON_H__


// Welche Versionen gibt es?
#define FELDVERSION    1
#define STRINGVERSION  2
#define ZEIGERVERSION  3


// Welche Version soll es sein?
//#define AKTUELLEVERSION  FELDVERSION
#define AKTUELLEVERSION  STRINGVERSION
//#define AKTUELLEVERSION  ZEIGERVERSION


#include <iostream>


#if AKTUELLEVERSION==FELDVERSION
#include <cstring>
#include <string>
#include <stdexcept>
#define LAENGENAMEN    50 // maximale Länge inkl. '\0'
#elif AKTUELLEVERSION==STRINGVERSION
#include <string>
#elif AKTUELLEVERSION==ZEIGERVERSION
#include <cstring>
#include <cstdio>
#else
#error "AKTUELLEVERSION nicht definiert"
#endif // AKTUELLEVERSION...
```

```
class Person
{
public:

#if AKTUELLEVERSION==FELDVERSION

  // Konstruktor
  Person( const char *vn="",
          const char *nn="",
          unsigned short nLenze=0u
          )
  {
    // setze() macht eh das Selbe:
    setze( vn, nn, nLenze );
  }

  // (kein Destruktor nötig)

  // setzt den Inhalt komplett neu:
  void setze( const char *vn="",
              const char *nn="",
              unsigned short nLenze=0u
              )
  {
    // Länge der Namen prüfen, ggf. Ausnahme werfen:
    if( strlen( vn )>=LAENGENAMEN
        ||
        strlen( nn )>=LAENGENAMEN
        )
    {
      // Namen zu lang, abbrechen!
      throw std::out_of_range( "Name zu lang in Person::setze()" );
    }

    // ok, beide Namen können kopiert werden:
    strcpy( vorname, vn );
    strcpy( nachname, nn );

    alter = nLenze;
  }


#elif AKTUELLEVERSION==STRINGVERSION

  // Konstruktor
  Person( const char *vn="",
          const char *nn="",
          unsigned short nLenze=0u
          )
    : vorname( vn ),
      nachname( nn ),
      alter( nLenze )
  {
  }
```

```
  // (kein Destruktor nötig)

  // setzt den Inhalt komplett neu:
  void setze( const char *vn="",
              const char *nn="",
              unsigned short nLenze=0u
              )
    {
      vorname = vn;  // string läßt sich mit Zuweisung kopieren
      nachname = nn;
      alter = nLenze;
    }

  bool operator==( const Person &rS ) const
    {
      return nachname==rS.nachname;
    }

#elif AKTUELLEVERSION==ZEIGERVERSION

  // Konstruktor
  Person( const char *vn="",
          const char *nn="",
          unsigned short nLenze=0u
          )
  {
    // setze() macht eh das Selbe, allerdings müssen vorname und
    // nachname zu NULL gesetzt werden, weil sonst delete[] mit deren
    // zufälligen bisherigem Inhalt aufgerufen wird, ein delete[]NULL
    // dagegen schadet nichts:
    vorname = NULL;
    nachname = NULL;
    setze( vn, nn, nLenze );
  }

  // Destruktor
  virtual ~Person()
  {
    // im ctor oder setze() allokierten Speicher wieder freigeben:
    delete[] vorname;  vorname = NULL;
    delete[] nachname; nachname = NULL;
  }

  // setzt den Inhalt komplett neu:
  void setze( const char *vn="",
              const char *nn="",
              unsigned short nLenze=0u
              )
  {
    // Beide bisherigen Namen freigeben:
    delete[] vorname;  vorname = NULL;
    delete[] nachname; nachname = NULL;
```

```
    // Mit passender Länge neu allokieren:
    vorname = new char[strlen( vn ) + 1];
    nachname = new char[strlen( nn ) + 1];


    // new klappt immer, beide Namen können kopiert werden:
    strcpy( vorname, vn );
    strcpy( nachname, nn );

    alter = nLenze;
  }

  // Überschreiben der Funktion operator=() noch nicht
  // aktiviert; siehe Beschreibung der Musterlösung

//  // Zuweisung Person = Person
//  Person & operator=( const Person &rechteSeite )
//  {
//    // Beide bisherigen Namen freigeben:
//    delete[] vorname;  vorname = NULL;
//    delete[] nachname; nachname = NULL;
//
//    // Anhand der rechten Seite der Zuweisung neu allokieren
//    // und kopieren:
//    setze( rechteSeite.vorname,
//           rechteSeite.nachname,
//           rechteSeite.alter
//           );
//
//    return *this;
//  }

#else
#error "AKTUELLEVERSION nicht definiert"
#endif // AKTUELLEVERSION...

  // Ausgabe einer Person:
  void zeige() const
  {
    // Ausgabe nach cout klappt mit char[], char* und string:
    std::cout << vorname << " "
              << nachname << ", "
              << alter
              << std::endl;
  }

private:

  // Die Namen:

#if AKTUELLEVERSION==FELDVERSION

  char          vorname[LAENGENAMEN];  // Feste Länge!
  char          nachname[LAENGENAMEN];
```

```
#elif AKTUELLEVERSION==STRINGVERSION

  std::string   vorname;  // wird automatisch verwaltet
  std::string   nachname;

#elif AKTUELLEVERSION==ZEIGERVERSION

  char         *vorname;  // nur Zeiger, rechtzeitig allokieren
  char         *nachname; // und wieder freigeben!

#else
#error "AKTUELLEVERSION nicht definiert"
#endif // AKTUELLEVERSION...

  // Das Alter:
  unsigned short alter;  // nicht mehr als 65535 Jahre möglich


}; // class Person...

#endif // ifdef __PERSON_H__
```

Bei der Verwendung von char-Feldern treten keine Probleme auf (abgesehen von der üblichen Platzverschwendung und der harten maximalen Länge der Namen natürlich).

Nach dem Initialisieren haben die beiden Objekte `a` und `b` die gewünschten Werte, nach der Zuweisung sind sie gleich, und nach dem erneuten Setzen hat `b` den gesetzten Wert:

```
klaus@lap2:~/skript_cpp > g++ -Wall test_person.cpp -o test_person
klaus@lap2:~/skript_cpp > test_person
(definieren)
Karl Moik, 72
Otto Waalkes, 55
(zuweisen)
Otto Waalkes, 55
Otto Waalkes, 55
(b setzen)
Otto Waalkes, 55
Eddi Stotter, 88
```

In der Version mit C++-strings funktioniert die Klasse ebenfalls wie erwartet, und erzeugt exakt dieselbe Ausgabe wie die Version mit Feldern (siehe oben).

Die Version mit Zeigern verhält sich dagegen anders (Beispiel unter Linux getestet, auf anderen Systemen kann es etwas anders aussehen, aber jedenfalls nicht wie bei den obigen Versionen):

```
klaus@lap2:~/skript_cpp > g++ -Wall test_person.cpp -o test_person
```

```
klaus@lap2:~/skript_cpp > test_person
(definieren)
Karl Moik, 72
Otto Waalkes, 55
(zuweisen)
Otto Waalkes, 55
Otto Waalkes, 55
(b setzen)
Eddi Stotter, 55
Eddi Stotter, 88
Segmentation fault
```

Daran sind zwei Punkte sehr befremdlich:

1. Nach der Zuweisung sind die Objekte `a` und `b` zwar erwartungsgemäß gleich (`Otto Waalkes, 55`), aber nach dem Setzen von `b` auf einen neuen Wert haben sich Vor- und Nachname in `a` ebenfalls geändert!
2. Am Ende des Programms (Ende von `main()`) stürzt das Programm ab (`Segmentation fault`).

Was ist passiert? Das Problem liegt daran, daß mit der Zuweisung `a=b` in `main()` alles, was innerhalb der Klasse definiert ist (`vorname`, `nachname`, `alter`) exakt von `b` nach `a` kopiert wird. Das bedeutet, daß auch die Zeiger umkopiert werden. Dadurch werden beide Zeiger `a.vorname` und `a.nachname` durch `b.vorname` und `b.nachname` ersetzt. Dagegen werden die eigentlichen Namen (also die Zeichen in den mit `new` allokierten Speicherbereichen) nicht kopiert.

Weil jetzt die Zeiger in `a` auf dieselben Stelle im Speicher zeigen wie die in `b`, wirkt sich eine Änderung der Namen in `b` auch auf `a` aus. (Mit dem Neusetzen der Namen in `b` wird freigegeben und neu allokiert; theoretisch könnte dabei ein Speicher an einer anderen Stelle im Speicher reserviert werden; dann würden die Zeiger in `a` nicht auf denselben Speicher zeigen, sondern auf bereits freigegebenen Speicher. Das scheint hier aber nicht der Fall zu sein, weil die neuen Namen nicht wesentlich länger sind als die alten und deshalb der mit `delete` freigegebene Speicher mit dem `new` gleich wieder reserviert wird).

Dies erklärt die Tatsache, daß sich eine Änderung der Namen in `b` auf die Namen von `a` auswirkt. Dieser Effekt ist sicher nicht erwünscht.

Aber warum stürzt das Programm zum Schluß ab? Dies liegt daran, daß am Ende der Lebensdauer von beiden Variablen jeweils ihr Destruktor aufgerufen wird, zuerst der von `b`. Hierin wird der Speicher für die beiden Namen freigegeben; danach wird der Destruktor von `a` aufgerufen. Weil durch die Zuweisung aber in `a` dieselben Zeiger stehen wie in `b`, wird versucht, den bereits im ersten Destruktoraufruf freigegebenen Speicher nochmals freizugeben. Dies führt zum Absturz. Andererseits kann der vor der Zuweisung für `a` allokierte Speicher nie vom Programm freigegeben werden, weil kein Zeiger mehr darauf existiert.

Nachdem die Probleme durch die vom Compiler unsinnig durchgeführte Zuweisung verursacht werden (auch wenn sie erst später in Erscheinung treten), liegt die Lösung darin, die Zuweisung zu verbessern.

Dazu kann man sich eine Funktion `operator=()` definieren, mit der man definiert, was bei einer Zuweisung genau passieren soll (siehe auch im C++-Skript den Abschnitt **Überladen von Operatoren**).

In der Musterlösung `Person.h` ist dies bereits (auskommentiert) enthalten. Wenn man die Kommentarzeichen vor der Funktion `operator()` entfernt, erhält man eine stabile Programmversion, die wie erwartet funktioniert und exakt die selbe Ausgabe ergibt wie die beiden ersten Versionen (mit Feldern beziehungsweise `string`).

Anstatt einfach die Zeiger zu kopieren, wird der Speicher der linken Seite freigegeben, in passender Länge neu allokiert, und dann werden die Zeichen der Namen mit `strcpy()` umkopiert.

8 C++-Liste von Personen

8.1 Aufgabe: C++-Liste von Personen

Es ist eine einfach verkettete Liste zu bauen mit den Eigenschaften wie in der Aufgabe **4 Einfach verkettete Liste** beschrieben; allerdings in C++.

Für alle Klassen sind also sinnvolle Konstruktoren, Desktruktoren etc. vorzusehen; und die bisherigen globalen Funktionen beispielsweise zum Einfügen sind jetzt als Methoden zu formulieren.

Spätester Abgabetermin ist Mittwoch, der 15.12.2004. Empfohlen ist eine Bearbeitung bis 08.12.2004.

8.2 Lösung: C++-Liste von Personen

Für die Person wird die Klasse `Person` aus den vorigen Aufgaben verwendet (in der Version mit `std::string` für die Namen, weil sonst der fehlende copy-Konstruktor Ärger macht):

```
// Time-stamp: "15.12.04 17:37 Person.h klaus@wachtler.de"
//
// Deklarationen (und Definitionen) für die class Person aus der
// C++-Übung A
//
// Musterlösung 08.12.2002 Klaus Wachtler
//
// 15.12.2004 kw   operator== ergänzt für Aufgabe 8

#ifndef __PERSON_H__
#define __PERSON_H__


// Welche Versionen gibt es?
#define FELDVERSION    1
#define STRINGVERSION  2
#define ZEIGERVERSION  3


// Welche Version soll es sein?
//#define AKTUELLEVERSION  FELDVERSION
#define AKTUELLEVERSION  STRINGVERSION
//#define AKTUELLEVERSION  ZEIGERVERSION


#include <iostream>
```

```
#if AKTUELLEVERSION==FELDVERSION
#include <cstring>
#include <string>
#include <stdexcept>
#define LAENGENAMEN    50 // maximale Länge inkl. '\0'
#elif AKTUELLEVERSION==STRINGVERSION
#include <string>
#elif AKTUELLEVERSION==ZEIGERVERSION
#include <cstring>
#include <cstdio>
#else
#error "AKTUELLEVERSION nicht definiert"
#endif // AKTUELLEVERSION...

class Person
{
public:

#if AKTUELLEVERSION==FELDVERSION

  // Konstruktor
  Person( const char *vn="",
          const char *nn="",
          unsigned short nLenze=0u
          )
  {
    // setze() macht eh das Selbe:
    setze( vn, nn, nLenze );
  }

  // (kein Destruktor nötig)

  // setzt den Inhalt komplett neu:
  void setze( const char *vn="",
              const char *nn="",
              unsigned short nLenze=0u
              )
  {
    // Länge der Namen prüfen, ggf. Ausnahme werfen:
    if( strlen( vn )>=LAENGENAMEN
        ||
        strlen( nn )>=LAENGENAMEN
        )
    {
      // Namen zu lang, abbrechen!
      throw std::out_of_range( "Name zu lang in Person::setze()" );
    }

    // ok, beide Namen können kopiert werden:
    strcpy( vorname, vn );
    strcpy( nachname, nn );

    alter = nLenze;
  }
```

```
#elif AKTUELLEVERSION==STRINGVERSION

  // Konstruktor
  Person( const char *vn="",
          const char *nn="",
          unsigned short nLenze=0u
          )
    : vorname( vn ),
      nachname( nn ),
      alter( nLenze )
  {
  }

  // (kein Destruktor nötig)

  // setzt den Inhalt komplett neu:
  void setze( const char *vn="",
              const char *nn="",
              unsigned short nLenze=0u
              )
    {
      vorname = vn;  // string läßt sich mit Zuweisung kopieren
      nachname = nn;
      alter = nLenze;
    }

  bool operator==( const Person &rS ) const
    {
      return nachname==rS.nachname;
    }

#elif AKTUELLEVERSION==ZEIGERVERSION

  // Konstruktor
  Person( const char *vn="",
          const char *nn="",
          unsigned short nLenze=0u
          )
  {
    // setze() macht eh das Selbe, allerdings müssen vorname und
    // nachname zu NULL gesetzt werden, weil sonst delete[] mit deren
    // zufälligen bisherigem Inhalt aufgerufen wird, ein delete[]NULL
    // dagegen schadet nichts:
    vorname = NULL;
    nachname = NULL;
    setze( vn, nn, nLenze );
  }

  // Destruktor
  virtual ~Person()
  {
    // im ctor oder setze() allokierten Speicher wieder freigeben:
```

```
    delete[] vorname;  vorname = NULL;
    delete[] nachname; nachname = NULL;
  }

  // setzt den Inhalt komplett neu:
  void setze( const char *vn="",
              const char *nn="",
              unsigned short nLenze=0u
              )
  {
    // Beide bisherigen Namen freigeben:
    delete[] vorname;  vorname = NULL;
    delete[] nachname; nachname = NULL;

    // Mit passender Länge neu allokieren:
    vorname = new char[strlen( vn ) + 1];
    nachname = new char[strlen( nn ) + 1];


    // new klappt immer, beide Namen können kopiert werden:
    strcpy( vorname, vn );
    strcpy( nachname, nn );

    alter = nLenze;
  }

  // Überschreiben der Funktion operator=() noch nicht
  // aktiviert; siehe Beschreibung der Musterlösung

//  // Zuweisung Person = Person
//  Person & operator=( const Person &rechteSeite )
//  {
//    // Beide bisherigen Namen freigeben:
//    delete[] vorname;  vorname = NULL;
//    delete[] nachname; nachname = NULL;
//
//    // Anhand der rechten Seite der Zuweisung neu allokieren
//    // und kopieren:
//    setze( rechteSeite.vorname,
//           rechteSeite.nachname,
//           rechteSeite.alter
//           );
//
//    return *this;
//  }

#else
#error "AKTUELLEVERSION nicht definiert"
#endif // AKTUELLEVERSION...

  // Ausgabe einer Person:
  void zeige() const
  {
    // Ausgabe nach cout klappt mit char[], char* und string:
```

```
    std::cout << vorname << " "
              << nachname << ", "
              << alter
              << std::endl;
  }

private:

  // Die Namen:

#if AKTUELLEVERSION==FELDVERSION

  char          vorname[LAENGENAMEN];  // Feste Länge!
  char          nachname[LAENGENAMEN];

#elif AKTUELLEVERSION==STRINGVERSION

  std::string   vorname;  // wird automatisch verwaltet
  std::string   nachname;

#elif AKTUELLEVERSION==ZEIGERVERSION

  char         *vorname;  // nur Zeiger, rechtzeitig allokieren
  char         *nachname; // und wieder freigeben!

#else
#error "AKTUELLEVERSION nicht definiert"
#endif // AKTUELLEVERSION...

  // Das Alter:
  unsigned short alter;  // nicht mehr als 65535 Jahre möglich


}; // class Person...

#endif // ifdef __PERSON_H__
```

Für die Suche nach einem Element wurde der `operator==` ergänzt.

Die Liste sieht dann so aus:

```
// Time-stamp: "15.12.04 17:56 ListePerson.h klaus@wachtler.de"
//
// Musterlösung zur C-Übung 8:
//
// "C++Liste von Personen"
//
// Wie Aufgabe 4, aber in C++ statt C.
//
// 15.12.2004 kw     Als Musterlösung beschrieben
//
```

```
#ifndef _LISTE_PERSON_H_
#define _LISTE_PERSON_H_

#include <cstdio>

#include "Person.h"


class ListePerson
{
public:

  ListePerson()
    : listenAnfang( NULL )
  {
  }

  ~ListePerson()
  {
    delete listenAnfang;
  }

  void einfuegen_vorne( const Person &person )
    {
      listenAnfang = new ListenElement( person, listenAnfang );
    }

  void einfuegen_hinten( const Person &person )
    {
      // Zeiger ab LA durch die Liste laufen lassen, bis ein
      // next gefunden wird, welches NULL ist:
      ListenElement  **ptr_lanf_oder_next = &listenAnfang;
      while( *ptr_lanf_oder_next )
      {
        ptr_lanf_oder_next = &((*ptr_lanf_oder_next)->next);
      }
      // Wenn es Elemente gibt, dann zeigt lanf_p jetzt auf das letzte
      // next der Liste (also das mit dem Wert NULL).
      //
      // Wenn die Liste leer ist, zeigt lanf_p auf den Listenanfang
      // wie nach dem Initialisieren.


      // hier einfügen:
      *ptr_lanf_oder_next = new ListenElement( person, *ptr_lanf_oder_next );
    }

  // Iteriert über alle Elemente, ruft für jedes Element die call
  // back-Funktion auf, und liefert die Anzahl der besuchten
  // Elemente zurück.
  int iteriere(  void(*callback)( const Person& ) )
    {
      int anzahl = 0;
      ListenElement   *aktLE = listenAnfang;
```

```
    while( aktLE )
    {
      callback( *( aktLE->person ) );
      anzahl++;
      aktLE = aktLE->next;
    }
    return anzahl;
  }

// Löscht aus einer Liste alle Einträge, die der übergebenen gleich
// sind:
void loesche( const Person &person )
  {
    // alle Einträge abklappern, und mit dem übergebenen Namen
    // vergleichen:
    ListenElement    **ptr_lanf_oder_next = &listenAnfang;
    while( *ptr_lanf_oder_next )
    {
      if( *( (*ptr_lanf_oder_next)->person ) == person )
      {
        // Nachname gleich, also löschen:

        // Rest der Liste merken
        ListenElement *alter_next = (*ptr_lanf_oder_next)->next;

        // Bei dem zu löschenden Element den Nachfolger zu NULL
        // setzen, damit beim folgenden delete nicht die ganze
        // restliche Liste freiegegeben wird:
        (*ptr_lanf_oder_next)->next = NULL;
        delete *ptr_lanf_oder_next;         // Listenelement freigeben
        *ptr_lanf_oder_next = alter_next;   // Rest der Liste
                                            // wieder einhängen
      }
      else
      {
        // aktuelles Element nicht zu löschen, also eins
        // weitergehen:
        ptr_lanf_oder_next = &((*ptr_lanf_oder_next)->next);
      }
    }
  }

// Sucht anhand von operator== nach einer Person und liefert
// einen Zeiger darauf oder NULL:
const Person *suche( const Person &person )
{
    ListenElement   *aktLE = listenAnfang;
    while( aktLE )
    {
      if( *( aktLE->person ) == person )
      {
        // Element gefunden!
        return aktLE->person;
      }
```

```
        // eins weiter gehen:
        aktLE = aktLE->next;
      }
      // nix gefunden, also NULL liefern:
      return NULL;
    }

  // Gibt eine Liste frei (und setzt den Listenanfang zu NULL):
  void loesche()
  {
    delete listenAnfang;
    listenAnfang = NULL;
  }

  /*+++++++++++++++++++++++++++++++++++++++++++++++++++++++*/

 private:

  // Ein Listenelement (privat in ListenElement, weil es niemand
  // außerhalb braucht):
  class ListenElement
    {
    public:

      ListenElement( const Person &person, ListenElement *next = NULL )
        : person( new Person( person ) ),
          next( next )
        {
        }

      ~ListenElement()
      {
        delete person;
        delete next;   // gibt rekursiv die ganze Liste frei
      }


      const Person          *person;  // Nutzdaten
      ListenElement         *next;    // Zeiger auf Nachfolger
    };

  ListenElement   *listenAnfang;

}; // class ListePerson
#endif /* ifndef _LISTE_PERSON_H_ */
```

Das Testprogramm:

```
// Time-stamp: "15.12.04 17:30 testListePerson.cpp klaus@wachtler.de"
//
// Testprogramm für ListePerson.h (Musterlösung Aufgabe 8)
```

```
//
// 15.12.2004 kw    Als Musterlösung erstellt

#include <iostream>

#include "Person.h"
#include "ListePerson.h"


// Gibt eine Person aus:
void zeigeLEPerson( const Person &person )
{
  person.zeige();
}


int main( int nargs, char **args )
{
  ListePerson  liste1;
  ListePerson  liste2;

  // Liste aufbauen:
  liste1.einfuegen_vorne( Person( "Klaus", "Wachtler", 42 ) );
  liste1.einfuegen_vorne( Person( "Otto", "Waalkes", 58 ) );
  liste1.einfuegen_vorne( Person( "Inge", "Meisel", 128 ) );
  liste1.einfuegen_hinten( Person( "Angie", "Merkel", 72 ) );
  liste1.einfuegen_hinten( Person( "Zorro", "Schatten", 22 ) );
  liste1.einfuegen_hinten( Person( "Dschingis", "Khan", 321 ) );

  // Liste ausgeben:
  std::cout << "erste Liste:" << std::endl;
  liste1.iteriere( zeigeLEPerson );

  // Zweite Liste aufbauen:
  std::cout << "zweite Liste:" << std::endl;
  liste2.einfuegen_hinten( Person( "Frieda", "Meier", 45 ) );
  liste2.einfuegen_vorne( Person( "Esmeralda", "Meier", 23 ) );
  liste2.einfuegen_vorne( Person( "Zumpfi", "Dumpfbacke", 12 ) );

  /* Liste ausgeben:
   */
  liste2.iteriere( zeigeLEPerson );

  std::cout << "nach Löschen:" << std::endl;
  //liste2.loesche( Person( "Frieda", "Meier", 45 ) );
  liste2.loesche( Person( "Zumpfi", "Dumpfbacke", 12 ) );
  liste2.iteriere( zeigeLEPerson );

  const Person *gefundene_person
    = liste1.suche( Person( "Kasimir", "Waalkes", 0 ) );
  if( gefundene_person )
  {
    std::cout << "gefunden:" << std::endl;
    gefundene_person->zeige();
```

```
  }
  else
  {
    std::cout << "nichts gefunden!" << std::endl;
  }

  // Listen löschen:
  liste1.loesche();
  liste2.loesche();

  std::cout << "nach dem Löschen:" << std::endl;
  liste1.iteriere( zeigeLEPerson );
  liste2.iteriere( zeigeLEPerson );

  return 0;
} /* main( int nargs, char **args ) */
```

9 Listentyp als Makro

9.1 Aufgabe: Listentyp als Makro

Die vorherige Aufgabe ist so umzubauen, daß mit wenig Aufwand Listen von verschiedenen Typen gebildet werden können.

Das Anwenderprogramm soll in etwa so aussehen:

```
#include <iostream>

#include "makroliste.h"

#include "Person.h"
//#include "Auto.h"
//#include "Student.h"

MACHLISTE( Person, PersonenListe );
//MACHLISTE( Auto, AutoListe );
//MACHLISTE( Student, StudentListe );


int main( int nargs, char **args )
{
  PersonenListe      perslist;
  //AutoListe          pkwlist;
  //StudentListe       studilist;

  perslist.einAnfang( Person( "Karl", "Meier", 89 ) );
  // ...

  return 0;
} // main( int nargs, char **args )
```

In der Headerdatei `makroliste.h` soll jetzt ein Makro `MACHLISTE` für den Präprozessor abgelegt werden, welches mit zwei Parametern aufgerufen wird: dem Namen eines existierenden Datentyps für Nutzdaten, und einem Namen für einen gewünschten Datentyp einer Liste der Nutzdaten.

Mit dem Makroaufruf `MACHLISTE( Person, PersonenListe )` soll das Makro zu etwas in der folgenden Art expandieren:

```
class PersonenListe
{
private:
   class Listenelement
   {
   public:
      Person *nutzdaten;
```

```
      Listenelement *next;
      ...
   };

public:
   void einAnfang( const Person *daten )
   {
      ...
   }
   ...

};
```

Analog würde ein Aufruf `MACHLISTE( Auto, AutoListe )` einen Datentyp für eine Liste von Autos erzeugen.

Spätester Abgabetermin ist Mittwoch, der 22.12.2004. Empfohlen ist eine Bearbeitung bis 15.12.2004.

9.2 Lösung: Listentyp als Makro

Für die Person wird die Klasse `Person` aus der vorigen Aufgabe verwendet und hier nicht nochmals aufgelistet.

Die Liste ist durch ein Makro ersetzt, welches bei jedem Aufruf eine ganze Klasse erzeugt:

```
// Time-stamp: "16.12.04 15:03 makroliste.h klaus@wachtler.de"
//
// Musterlösung zur C-Übung 9:
//
// "Listentyp als Makro"
//
// Wie Aufgabe 8, aber nicht direkt eine Klasse, sondern ein Makro,
// welches eine Klasse erzeugt.
//
// 15.12.2004 kw     Als Musterlösung erstellt (aus ListePerson.h)
//

#ifndef _MAKROLISTE_H_
#define _MAKROLISTE_H_

#include <cstdio>

#define MACHLISTE(DATENTYP,LISTENTYP)                                         \
class LISTENTYP                                                               \
{                                                                             \
public:                                                                       \
                                                                              \
  LISTENTYP()                                                                 \
```

```
  : listenAnfang( NULL )                                                    \
{                                                                           \
}                                                                           \
                                                                            \
~LISTENTYP()                                                                \
{                                                                           \
  delete listenAnfang;                                                      \
}                                                                           \
                                                                            \
void einfuegen_vorne( const DATENTYP &daten )                               \
  {                                                                         \
    listenAnfang = new ListenElement( daten, listenAnfang );                \
  }                                                                         \
                                                                            \
void einfuegen_hinten( const DATENTYP &daten )                              \
  {                                                                         \
    /* Zeiger ab LA durch die Liste laufen lassen, bis ein */               \
    /* next gefunden wird, welches NULL ist:               */               \
    ListenElement  **ptr_lanf_oder_next = &listenAnfang;                    \
    while( *ptr_lanf_oder_next )                                            \
    {                                                                       \
      ptr_lanf_oder_next = &((*ptr_lanf_oder_next)->next);                  \
    }                                                                       \
    /* Wenn es Elemente gibt, dann zeigt lanf_p jetzt auf das letzte */     \
    /* next der Liste (also das mit dem Wert NULL).                  */     \
    /*                                                               */     \
    /* Wenn die Liste leer ist, zeigt lanf_p auf den Listenanfang    */     \
    /* wie nach dem Initialisieren.                                  */     \
                                                                            \
                                                                            \
    /* hier einfügen: */                                                    \
    *ptr_lanf_oder_next = new ListenElement( daten, *ptr_lanf_oder_next );  \
  }                                                                         \
                                                                            \
/* Iteriert über alle Elemente, ruft für jedes Element die call */          \
/* back-Funktion auf, und liefert die Anzahl der besuchten      */          \
/* Elemente zurück.                                             */          \
int iteriere(  void(*callback)( const DATENTYP& ) )                         \
  {                                                                         \
    int anzahl = 0;                                                         \
    ListenElement   *aktLE = listenAnfang;                                  \
    while( aktLE )                                                          \
    {                                                                       \
      callback( *( aktLE->daten ) );                                        \
      anzahl++;                                                             \
      aktLE = aktLE->next;                                                  \
    }                                                                       \
    return anzahl;                                                          \
  }                                                                         \
                                                                            \
/* Löscht aus einer Liste alle Einträge, die der übergebenen gleich */      \
/* sind:                                                            */      \
void loesche( const DATENTYP &daten )                                       \
  {                                                                         \
```

```
    /* alle Einträge abklappern, und mit dem übergebenen Namen */            \
    /* vergleichen:                                             */            \
    ListenElement    **ptr_lanf_oder_next = &listenAnfang;                   \
    while( *ptr_lanf_oder_next )                                              \
    {                                                                         \
      if( *( (*ptr_lanf_oder_next)->daten ) == daten )                       \
      {                                                                       \
        /* Nachname gleich, also löschen: */                                 \
                                                                              \
        /* Rest der Liste merken */                                           \
        ListenElement *alter_next = (*ptr_lanf_oder_next)->next;             \
                                                                              \
        /* Bei dem zu löschenden Element den Nachfolger zu NULL */           \
        /* setzen, damit beim folgenden delete nicht die ganze  */           \
        /* restliche Liste freiegegeben wird:                   */           \
        (*ptr_lanf_oder_next)->next = NULL;                                  \
        delete *ptr_lanf_oder_next;         /* Listenelement freigeben */    \
        *ptr_lanf_oder_next = alter_next;   /* Rest der Liste          */    \
                                            /* wieder einhängen        */    \
      }                                                                       \
      else                                                                    \
      {                                                                       \
        /* aktuelles Element nicht zu löschen, also eins */                  \
        /* weitergehen:                                   */                  \
        ptr_lanf_oder_next = &((*ptr_lanf_oder_next)->next);                 \
      }                                                                       \
    }                                                                         \
  }                                                                           \
                                                                              \
/* Sucht anhand von operator== nach einem Element und liefert */             \
/* einen Zeiger darauf oder NULL:                              */             \
const DATENTYP *suche( const DATENTYP &daten )                               \
{                                                                             \
    ListenElement   *aktLE = listenAnfang;                                   \
    while( aktLE )                                                            \
    {                                                                         \
      if( *( aktLE->daten ) == daten )                                       \
      {                                                                       \
        /* Element gefunden! */                                               \
        return aktLE->daten;                                                  \
      }                                                                       \
                                                                              \
      /* eins weiter gehen: */                                                \
      aktLE = aktLE->next;                                                    \
    }                                                                         \
    /* nix gefunden, also NULL liefern: */                                    \
    return NULL;                                                              \
  }                                                                           \
                                                                              \
/* Gibt eine Liste frei (und setzt den Listenanfang zu NULL): */             \
void loesche()                                                                \
{                                                                             \
  delete listenAnfang;                                                        \
  listenAnfang = NULL;                                                        \
```

```
  }                                                                               \
                                                                                  \
  /*++++++++++++++++++++++++++++++++++++++++++++++++++++++++*/                    \
                                                                                  \
 private:                                                                         \
                                                                                  \
  /* Ein Listenelement (privat in ListenElement, weil es niemand */               \
  /* außerhalb braucht):                                         */               \
  class ListenElement                                                             \
    {                                                                             \
    public:                                                                       \
                                                                                  \
      ListenElement( const DATENTYP &daten, ListenElement *next = NULL )          \
        : daten( new DATENTYP( daten ) ),                                         \
          next( next )                                                            \
        {                                                                         \
        }                                                                         \
                                                                                  \
      ~ListenElement()                                                            \
      {                                                                           \
        delete daten;                                                             \
        delete next;   /* gibt rekursiv die ganze Liste frei */                   \
      }                                                                           \
                                                                                  \
      const DATENTYP          *daten;   /* Nutzdaten              */              \
      ListenElement           *next;    /* Zeiger auf Nachfolger  */              \
    };                                                                            \
                                                                                  \
  ListenElement   *listenAnfang;                                                  \
                                                                                  \
}; // class LISTENTYP


#endif /* ifndef _MAKROLISTE_H_ */
```

Das Testprogramm:

```
// Time-stamp: "15.12.04 17:58 testmakroliste.cpp klaus@wachtler.de"
//
//

//#include <cstdlib>
//#include <cstddef>
//#include <cstdio>
//#include <cstring>

#include <iostream>

#include "makroliste.h"

#include "Person.h"
```

```
//#include "Auto.h"
//#include "Student.h"

MACHLISTE( Person, ListePerson );

//MACHLISTE( Auto, AutoListe );
//MACHLISTE( Student, StudentListe );


// Gibt eine Person aus:
void zeigeLEPerson( const Person &person )
{
  person.zeige();
}


int main( int nargs, char **args )
{
  ListePerson  liste1;
  ListePerson  liste2;

  // Liste aufbauen:
  liste1.einfuegen_vorne( Person( "Klaus", "Wachtler", 42 ) );
  liste1.einfuegen_vorne( Person( "Otto", "Waalkes", 58 ) );
  liste1.einfuegen_vorne( Person( "Inge", "Meisel", 128 ) );
  liste1.einfuegen_hinten( Person( "Angie", "Merkel", 72 ) );
  liste1.einfuegen_hinten( Person( "Zorro", "Schatten", 22 ) );
  liste1.einfuegen_hinten( Person( "Dschingis", "Khan", 321 ) );

  // Liste ausgeben:
  std::cout << "erste Liste:" << std::endl;
  liste1.iteriere( zeigeLEPerson );

  // Zweite Liste aufbauen:
  std::cout << "zweite Liste:" << std::endl;
  liste2.einfuegen_hinten( Person( "Frieda", "Meier", 45 ) );
  liste2.einfuegen_vorne( Person( "Esmeralda", "Meier", 23 ) );
  liste2.einfuegen_vorne( Person( "Zumpfi", "Dumpfbacke", 12 ) );

  /* Liste ausgeben:
   */
  liste2.iteriere( zeigeLEPerson );

  std::cout << "nach Löschen:" << std::endl;
  //liste2.loesche( Person( "Frieda", "Meier", 45 ) );
  liste2.loesche( Person( "Zumpfi", "Dumpfbacke", 12 ) );
  liste2.iteriere( zeigeLEPerson );

  const Person *gefundene_person
    = liste1.suche( Person( "Kasimir", "Waalkes", 0 ) );
  if( gefundene_person )
  {
    std::cout << "gefunden:" << std::endl;
    gefundene_person->zeige();
```

```
  }
  else
  {
    std::cout << "nichts gefunden!" << std::endl;
  }

  // Listen löschen:
  liste1.loesche();
  liste2.loesche();

  std::cout << "nach dem Löschen:" << std::endl;
  liste1.iteriere( zeigeLEPerson );
  liste2.iteriere( zeigeLEPerson );

  return 0;
} // main( int nargs, char **args )
```

10 Liste als `template`

10.1 Aufgabe: Liste als `template`

Wie die vorige Aufgabe, aber als `template`.

Spätester Abgabetermin ist Mittwoch, der 05.01.2005.

11 Ausgabe verstehen

Im folgenden werden 3 Klassen und ein Testprogramm gegeben.

Es ist die Ausgabe des Programms zu verstehen; gegebenenfalls kann man das Testprogramm variieren.

Es ist keine Abgabe nötig; wichtig ist die Aufgabe aber trotzdem.

Die Quelltexte sind an der FH unter Linux zu finden unter `/home/staff/wachtler/uebung11`.

11.1 A.h

```
// Time-stamp: "05.01.05 08:00 A.h klaus@wachtler.de"
//
// definiert eine relativ sinnlose Klasse, die bei allen Aktionen
// ausgibt, was gerade passiert.

#ifndef _A_H_
#define _A_H_

#include <ostream>


class A
{
public:

  // Konstruktor
  A()
    : ersterwert( 0 )
  {
    std::cerr << "ctor A()" << std::endl;
  }

  A( int ersterwert )
    : ersterwert( ersterwert )
  {
    std::cerr << "ctor A( "
              << ersterwert
              << " )"
              << std::endl;
  }

  // copy-Konstruktor
  A( const A &rechteSeite )
    : ersterwert( rechteSeite.ersterwert )
  {
    std::cerr << "copy-ctor A( const A &rechteSeite )"
              << " rechteSeite.ersterwert = "
```

```
              << rechteSeite.ersterwert
              << std::endl;
  }

  // Destruktor
  virtual ~A()
  {
    std::cerr << "dtor ~A(), " << ersterwert << std::endl;
  }

  // Zuweisung A = A
  // Rückgabewert ist eine REFERENZ auf die linke Seite
  // (nicht const), und damit wieder als lvalue verwendbar.
  A & operator=( const A &rechteSeite )
  {
    std::cerr << "Zuweisung A=A, ls=" << ersterwert
              << ", rs=" << rechteSeite.ersterwert
              << std::endl;

    ersterwert = rechteSeite.ersterwert;

    return *this;
  }

  // Ausgabe auf einen ostream
  virtual void streamout( std::ostream & s ) const
  {
    s << ersterwert;
  }

  // Kopie von sich selbst allokieren
  virtual A *clone() const
  {
    return new A(*this);
  }

  // Leeres Objekt vom selben Typ allokieren
  virtual A *create() const
  {
    return new A();
  }

protected:

  // Membervariable (in den abgeleiteten Klassen kommen noch welche
  // dazu):
  int ersterwert;


}; // class A...


inline std::ostream &operator<<( std::ostream &s, const A &obj )
{
```

```
  obj.streamout( s );
  return s;
}

#endif /* ifndef _A_H_ */
```

11.2 B.h

```
// Time-stamp: "05.01.05 12:16 B.h klaus@wachtler.de"
//
// definiert eine relativ sinnlose Klasse, die bei allen Aktionen
// ausgibt, was gerade passiert.

#ifndef _B_H_
#define _B_H_

#include <ostream>

#include "A.h"


class B: public A
{
public:

  // Konstruktor
  B()
    : A(),
      zweiterwert( 0 )
  {
    std::cerr << "ctor B()" << std::endl;
  }

  B( int ersterwert, int zweiterwert )
    : A( ersterwert ),
      zweiterwert( zweiterwert )
  {
    std::cerr << "ctor B( "
              << ersterwert
              << ", "
              << zweiterwert
              << " )"
              << std::endl;
  }

  // copy-Konstruktor
  B( const B &rechteSeite )
    : A( rechteSeite.ersterwert ),
      zweiterwert( rechteSeite.zweiterwert )
  {
```

```
  std::cerr << "copy-ctor B( const B &rechteSeite )"
            << " rechteSeite.ersterwert = "
            << rechteSeite.ersterwert
            << ",  rechteSeite.zweiterwert = "
            << rechteSeite.zweiterwert
            << std::endl;
}

// Destruktor
virtual ~B()
{
  std::cerr << "dtor ~B()"
            << ", ersterwert=" << ersterwert
            << ", zweiterwert=" << zweiterwert
            << std::endl;
}

// Zuweisung B = B
// Rückgabewert ist eine REFERENZ auf die linke Seite
// (nicht const), und damit wieder als lvalue verwendbar.
B & operator=( const B &rechteSeite )
{
  std::cerr << "Zuweisung B=B"
            << ", ls.ersterwert=" << ersterwert
            << ", ls.zweiterwert=" << zweiterwert
            << ", rs.ersterwert=" << rechteSeite.ersterwert
            << ", rs.zweiterwert=" << rechteSeite.zweiterwert
            << std::endl;

  *(A*)this = *(A*)&rechteSeite;
  zweiterwert = rechteSeite.zweiterwert;

  return *this;
}

// Ausgabe auf einen ostream
virtual void streamout( std::ostream & s ) const
{
  A::streamout( s );
  s << zweiterwert;
}

// Kopie von sich selbst allokieren
virtual B *clone() const
{
  return new B(*this);
}

// Leeres Objekt vom selben Typ allokieren
virtual B *create() const
{
  return new B();
}
```

```
protected:

  // Membervariable
  int zweiterwert;


}; // class B...


// nicht nötig:
// inline std::ostream &operator<<( std::ostream &s, const B &obj )

#endif /* ifndef _B_H_ */
```

11.3 C.h

```
// Time-stamp: "05.01.05 12:27 C.h klaus@wachtler.de"
//
// definiert eine relativ sinnlose Klasse, die bei allen Aktionen
// ausgibt, was gerade passiert.

#ifndef _C_H_
#define _C_H_

#include <ostream>

#include "B.h"


class C: public B
{
public:

  // Konstruktor
  C()
    : B(),
      dritterwert( 0 )
  {
    std::cerr << "ctor C()" << std::endl;
  }

  C( int ersterwert, int zweiterwert, int dritterwert )
    : B( ersterwert, zweiterwert ),
      dritterwert( dritterwert )
  {
    std::cerr << "ctor C( "
              << ersterwert
              << ", "
              << zweiterwert
              << ", "
```

```
            << dritterwert
            << " )"
            << std::endl;
}

// copy-Konstruktor
C( const C &rechteSeite )
  : B( rechteSeite.ersterwert, rechteSeite.zweiterwert ),
    dritterwert( rechteSeite.dritterwert )
{
  std::cerr << "copy-ctor C( const C &rechteSeite )"
            << " rechteSeite.ersterwert = "
            << rechteSeite.ersterwert
            << ",  rechteSeite.zweiterwert = "
            << rechteSeite.zweiterwert
            << ",  rechteSeite.dritterwert = "
            << rechteSeite.dritterwert
            << std::endl;
}

// Destruktor
virtual ~C()
{
  std::cerr << "dtor ~C()"
            << ", ersterwert=" << ersterwert
            << ", zweiterwert=" << zweiterwert
            << ", dritterwert=" << dritterwert
            << std::endl;
}

// Zuweisung C = C
// Rückgabewert ist eine REFERENZ auf die linke Seite
// (nicht const), und damit wieder als lvalue verwendbar.
C & operator=( const C &rechteSeite )
{
  std::cerr << "Zuweisung C=C"
            << ", ls.ersterwert=" << ersterwert
            << ", ls.zweiterwert=" << zweiterwert
            << ", ls.dritterwert=" << dritterwert
            << ", rs.ersterwert=" << rechteSeite.ersterwert
            << ", rs.zweiterwert=" << rechteSeite.zweiterwert
            << ", rs.dritterwert=" << rechteSeite.dritterwert
            << std::endl;

  *(B*)this = *(B*)&rechteSeite;
  dritterwert = rechteSeite.dritterwert;

  return *this;
}

// Ausgabe auf einen ostream
virtual void streamout( std::ostream & s ) const
{
  B::streamout( s );
```

```
    s << dritterwert;
  }

  // Kopie von sich selbst allokieren
  virtual C *clone() const
  {
    return new C(*this);
  }

  // Leeres Objekt vom selben Typ allokieren
  virtual C *create() const
  {
    return new C();
  }

protected:

  // Membervariable
  int dritterwert;


}; // class C...


// nicht nötig:
// inline std::ostream &operator<<( std::ostream &s, const C &obj )

#endif /* ifndef _C_H_ */
```

11.4 testabc.cpp

```
// Time-stamp: "05.01.05 12:32 testabc.cpp klaus@wachtler.de"
//
// Testprogramm für die Klassen A, B und C

//#include <cstdlib>
//#include <cstddef>
//#include <cstdio>
//#include <cstring>
//#include <stdexcept>
//#include <fstream>

#include <iostream>

#include "A.h"
#include "B.h"
#include "C.h"

void zeigea_cbv( A dasA )
{
```

```
  std::cout << "zeigea_cbv: dasA = " << dasA << std::endl;
}

void zeigea_cbr( const A &dasA )
{
  std::cout << "zeigea_cbr: dasA = " << dasA << std::endl;
}

int main( int nargs, char **args )
{
  A einA;

  std::cout << "\nAusgabe:" << std::endl;

  std::cout << "\nTest von A" << std::endl;

  zeigea_cbv( einA );
  zeigea_cbr( einA );

  std::cout << "\nTest von B" << std::endl;

  B einB, einanderesB( 12, 15 );

  std::cout << "\nZuweisung:" << std::endl;
  einB = einanderesB;

  std::cout << "\nTest von C" << std::endl;

  C einC, einanderesC( 12, 15, 25 );

  std::cout << "\nZuweisung:" << std::endl;
  einC = einanderesC;

  zeigea_cbv( einC );
  zeigea_cbr( einC );

  std::cout << "\nZeigertest:" << std::endl;

  std::cout << "Erzeugen" << std::endl;
  C  *p_c = new C( 21, 22, 23 );

  std::cout << "Ausgeben" << std::endl;
  zeigea_cbv( *p_c );
  zeigea_cbr( *p_c );

  std::cout << "Löschen" << std::endl;

  std::cout << "\nok, fertig!" << std::endl;
  return 0;
} // main( int nargs, char **args )
```