Grundlagen der Informatik

Vorlesung

Felder und Strukturen

Prof. Dr.-Ing. Thomas Wiedemann
Fachgebiet Informatik / Mathematik

Überblick zur Vorlesung

- Verwendung von Vektoren und mehrdimensionalen Matrizen für mathematische Berechnungen und allgemeine Datenverarbeitungsaufgaben
- Definition und Anwendung von Strukturen und Feldern
- Einfache Berechnungs- und Analysefunktionen
- Suchen in Feldern und Strukturen
- Sortieren in Feldern und Strukturen

Mengenbasierte Datenverarbeitung

Einführung

- fast alle realen Prozesse beruhen auf, umfassen oder generieren Mengen von Daten oder Objekten – Einzelwerte sind eher die Ausnahme
  - Rechnungen setzen sich aus Rechnungspositionen zusammen
  - Messungen von physikalisch/technische Prozesse erzeugen große Mengen an Einzeldaten (z.B. Temperaturverlauf)
  - Bauteile werden durch eine Vielzahl kleinster Körper nachgebildet (FEM)
- die gegenwärtige Rechentechnik (von Neumann-Architektur) kann aber nur immer einen Wert bearbeiten
- die Verarbeitung von Mengen muss daher auf Algorithmen auf der Basis von Einzeloperationen zurückgeführt werden

Einführung in die Arbeit mit Vektoren und Matrizen

Lösung:

- Umsetzung von Vektoren und Matrizen in der EDV seit den 50er Jahren durch Bereitstellung mehrdimensionaler Datenfelder
- Grunddatentypen (Zahlen, Texte, ...) werden als ein großes Feld angelegt und können über einen einheitlichen Namen und einen Index angesprochen werden
- in fast allen Programmiersprachen werden eindimensionale Vektoren und mehrdimensionale Matrizen N-Ordnung gleichartig behandelt
- es sind auch Matrizen möglich mit N> 3 (z.B. 6 dimensional)

- **Die Verarbeitung von mehrdimensionalen Daten erfolgt in der Regel mittels Schleifen.**

Matrizen in Visual Basic

Deklaration von Vektoren und Matrizen

- die bisherige Variablendefinition wird durch **runde Klammern mit der Anzahl der zu definierenden Werte** ergänzt :

```
Dim  werte(100) as double : Rem Feld von 100 realen Zahlen
Dim  durchlaeufe(1000) as integer:  REM  1000 ganze Zahlen
Dim  texte(200) as string:  REM  Feld von 200 Texten
```

- Technischer Hinweis: VB beginnt bei 0 mit der Zählung und definiert damit real n+1 Elemente ! (andere Sprachen wie C beginnen ebenfalls bei 0 und enden aber schon bei N-1 !!)
- **alle bisher behandelten Zusatzangaben zur Sichtbarkeit (public) oder Gültigkeit (Global) sind analog anwendbar**

Speicherplatz und Deklaration mehrdimensionaler Feldern

Der notwendige Speicherplatz für Felder wächst mit deren Größe und Dimension:

Dim zaehler(100) as integer : // reserviert Speicherplatz für 100 Integer-Werte

= 100 * 2 Bytes = 200 Bytes

Dim summen(100) as double : // reserviert Speicherplatz für 100 Double-Werte

= 100 * 6 Bytes = 600 Bytes

(da VB bei 0 beginnt werden, real n+1 Elemente definiert !)

Mehrdimensionale Felder werden durch mehrere Indizes (mit Komma getrennt) definiert:

Dim zaehler(100,100) as integer : // reserviert Feld mit 100 x 100 ganzen Zahlen

= 100 * 100 * 2 Bytes = 20.000 Bytes = 20 KByte !

Dim summen(50,40,4) as double : // Feld mit 50 x 40 x 4 Double-Werten

= 50 * 40 * 4 * 6 Bytes = 48.000 Bytes = 48 KByte !

- Achtung: Bedingt durch das exponentielle Wachstum der Elementezahl können mehrdimensionale Felder sehr große Speichermengen erfordern. Bei professionellen Systemen kommen dann Spezialverfahren zur Reduzierung der Datenmenge zum Einsatz (Unterdrückung leerer Zellen, zeilenweise Verarbeitung ...)

Zugriff auf Felder und Manipulation von Feldzellen

- Feldinhalte werden durch Angabe des Feldnamens und des jeweiligen Indexwertes adressiert und damit analog zu den anderen Variablen verwendet

```
Dim a(100) as integer
a(4) = 10: a(5) = 13 : a(6) = 15
Msgbox   a(4)  :  Rem Anzeige des 4. Feldes
```

- zur Angabe des Indexwertes können auch Variable (Typ Integer) verwendet werden

```
Dim  i as integer
i = 4   : Rem Setze i auf 4
a(i)=10  a(i+1) = 13 : a(i+2) = 15  : Rem relative Adressierung über i
```

- Durch eine Veränderung der Indexvariablen über eine Schleife können ALLE Elemente eines Feldes sehr einfach adressiert werden :

```
Dim  a(100) as integer
for i=1 to 100
        a(i) = 10 : Rem  setzt alle 100 Felder auf 10
next i
```

Zugriff und Manipulation von mehrdimensionalen Feldern

- Feldinhalte werden durch Angabe des Feldnamens und **aller Indexwerte** adressiert und damit analog zu anderen Variablen verwendet

```
a(10,4) = 10  :  Msgbox a(10,4)
```

- Bei mehrdimensionalen Feldern müssen entsprechend viele einzelne Indexvariablen mit **geschachtelten Schleifen** über alle Kombinationen hochgezählt werden :

```
Rem  Berechnung und Ausgabe eines 2-dimensionalen Feldes
Rem  (es gibt KEINE direkte Funktion dafür)
Dim  i as integer, j as integer : Rem  Definition der Indizes
Dim  b(10,20) as double   : Rem Das 2-dimens. Feld
   for i=1 to 10    : Rem Start der äußeren Schleife
    for j=1 to 20  : Rem Start der inneren  Schleife
            b(i, j)  = i*j  :  Rem - einen Testwert setzen ...
            Debug.print b(i ,j)
    next j
  next i
```

Praktische Anwendung von Matrizen

Abbildung von Datenmengen

- je nach Struktur der Datenmengen sind diese in Vektoren oder mehrdimensionalen Matrizen abbildbar
- Den Dimensionen der Matrizen entsprechen die Ordnungsmerkmale der Daten
  - bei Lohnzahlung z.B. Zeilenindex = Personalnr Spaltenindex = Monatsumsatz
  - bei Messungen z.B. Zeilenindex = Meßstation Spaltenindex = Zeitpunkt

Bisher vorgestellte Matrizendefinition

- erlaubt nur homogene Daten (gleicher Datentyp aller Werte)
- für viele praktische Aufgabenstellungen sind jedoch gleichzeitig verschiedene Datentypen notwendig – z.B. Kundendaten : Texte (Namen, Adressen) , Zahlen (Alter, Umsatz, PLZ) , Datum (Geburtstag, Kunde_seit) , ...
- Lösung 1: für jeden Datentyp muß eine getrennte Matrix definiert werden :

  Dim Kunden_Texte(10000) as string, Kunden_Zahlen(1000) as double , ...

bessere Alternative: Definition heterogener Datenstrukturen **(-> STRUKTUREN)** und Verwendung dieser Strukturen innerhalb einer Matrix (siehe Folgeseiten)

- Alle nachfolgend vorgestellten Algorithmen sind prinzipiell sowohl auf homogene wie auch auf heterogen Felder anwendbar !

Definition Strukturen

- **eine Struktur beschreibt eine Zusammenfassung von verschiedenen, beliebigen Datentypen:**
  - Adreßdaten (Name, Strasse, PLZ, Ort, Telefon ...)
  - Rechnung (Kundendaten, Bestelldaten, Rechnungsdaten)
  - Musiktitel (Titelname, Interpret, Zeitdauer, Position auf der CD-ROM, ...)
- **und stellt diese unter einem neuen Namen als abgeleiteten Datentyp zur Verfügung (sogen. Verbundtyp)**

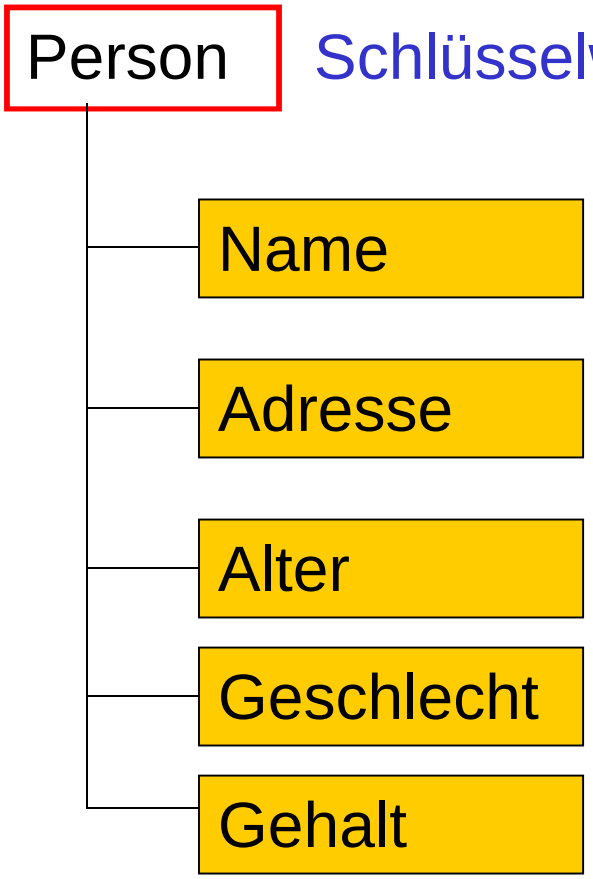
Grundlegende Regeln:

- innerhalb einer Struktur können **alle Grunddatentypen** (Zahlen, Zeichenketten, auch Arrays, Objekte) verwendet werden
- auch die Verwendung anderer Strukturen ist zulässig (nur der rekursive Aufruf der eigenen Struktur nicht)
- die **Verwendung von Datenstrukturen** erfolgt analog zur Verwendung der Grunddatentypen - es sind auch Arrays mit Pointern auf Strukturen in der Praxis sehr häufig im Einsatz

Interna (für alle Wißbegierigen)

- eine Struktur wird als Speicherblock verstanden
- der Name bzw. Pointer verweist auf die erste Speicherzelle
- über einen Offset (Verschiebung in Bytes) wird ausgehend von der Anfangsadresse auf die anderen Daten zugegriffen (=sehr gut unterstützt durch entspr. Prozessorbefehle = genau 1 Assemblerbefehl)

Aufbau und Deklaration einer Struktur in VB



Schlüsselwörter für Strukturen: **type** ... **end type**

```
type Person
    Name as string
    Adresse as string
    Alter as integer
    weiblich as boolean
    Gehalt as double
end type
```

Person ist der Strukturname
(und damit ein neuer Datentyp)

Name, Alter und ... sind Komponentennamen.

Definition und Zugriff auf die Daten von Strukturen

Nach der Deklaration von Strukturtypen können diese zur Definition von Variablen eingesetzt werden :

Deklaration :

```
type  Person
        Name as string
        Alter as integer : Rem ...
end type
```

Definition : Dim student as Person ,prof as Person

Zugriff : erfolgt durch Angabe des Variablennamens und dem Komponentennamen durch Punkt getrennt :

student.Name = “Jens Meier“ : student.alter = 22

prof.name=“Müller“ : prof.alter = getalter()

altersdifferenz = prof.alter - student.alter : ‘Normale Rechnung

Felder von Strukturen

Analog zu normalen Datentypen können Strukturen auch als Felder definiert werden :

Definition : Dim student(10000) as Person

Dim profs(600) as Person

Zugriff : erfolgt durch Angabe des Variablennamens, des Index in Klammern und dem Komponentennamen durch Punkt getrennt:

student(11).Name = “Jens Meier“ : student(11).alter = 22

student(12).Name = “Anja Michel “ : student(12).alter = 19

profs(55).name=“Müller“ : prof(55).alter = getalter():

Für Schreibfaule (und Performance-Süchtige) gibt es eine Vereinfachung mit **with** .... **with end** , welches nur einmal die Basisvariable erfordert:

```
with   student(56)
          .name=“Tom Meier“  :   .alter=23 :  .weiblich = False
end with
```

Basisangabe kann hier entfallen !

Allgemeine Operationen auf Feldern und Strukturfeldern

- Alle nachfolgend vorgestellten Algorithmen sind prinzipiell auf Feldern von einfachen und Strukturtypen zulässig. Es differiert nur der konkrete Zugriff ! Auch bei der Anzahl der Dimensionen können mehrdimensionalen Fälle analog abgeleitet werden!

Analyse

- Bestimmung vom Grenzwerten (Min/Max)
- Mathematische Berechnungen (Statistik, Matrizenberechnungen)
- Suche nach Werten
  - Frage nach der Existenz von Werten
  - Ermittlung der Position innerhalb des Feldes mit dem Ziel des Zugriffs auf damit zusätzlich verbundene Funktionen (z.B. Telefonbuch)

Verarbeitung

- Sortieren von Feldern
  - Beschleunigung von Suchprozessen
  - Aufbau von Sequenzen und Hierarchien
- Verdichtung von Daten (Zusammenfassung auf kleinere Zeitintervalle, z.B. Bestimmung des Monatsmittels aus den Tagesangaben ...)

Bestimmung vom Grenzwerten (Min/Max)

Prinzipieller Algorithmus:

1. **Setze Grenzwertvariable und Grenzwertindex auf das erste Feldelement**
2. **Gehe zum nächsten Feldelement**
3. **Falls dieses Element einen besseren Grenzwert darstellt, setze dessen Wert und Index als neue Grenzwertvariable und Grenzwertindex**
4. **Gehe zu 2. Falls noch weitere Feldelemente vorhanden sind**

```
Dim a(100) As Double, Start, i  As Integer
Rem  a soll beliebige Werte enthalten
Start = 1
 For i = Start To 100
    a(i) = (i - 50) * (i - 50)
  Next i
Maxwert = a(Start): MaxId = Start: Minwert = a(Start): MinId = Start
For i = Start To 100
  If a(i) > Maxwert Then Maxwert = a(i): MaxId = i:   ' Maximumsuche
  If a(i) <= Minwert Then Minwert = a(i):  MinId = i:  ' Minimumsuche
Next i
```

Suche nach Werten

Prinzipieller Algorithmus einer sequentiellen Suche :

```
Dim a(100) as double, Start, i  as integer, PositionsID  as integer
Rem  a soll beliebige Werte enthalten
Start=1 :  Suchwert = 56: PositionsID = -1:
Do
 if PositionsID>0 then  Start = PositionsID +1 :  PositionsID = -1:
 for i=Start to 100
  if a(i) = Suchwert then PositionsID = i  : break : REM Schleife ABBRECHEN !
next  i
  If PositionsID >=0 then  Msgbox "Wert gefunden bei Zelle" & str(PositionsID)
While PositionsID > 0
```

Achtung: Bei float/double ist i.d.R. ein Vergleich mittels Intervall günstiger :

```
if ( abs( a(i)  – Suchwert ) < eps)  ...
```

Aufwand: Minimal = 1 Maximal = N **im Mittel = n / 2 Vergleiche**

Bei großem N ist diese Methode ungünstig !!!

Suche nach Werten mittels Binärsuche

Bei großem N ist die sequentielle Suche zu langsam :

Alternative: Suche in einem sortierten Feld (Sortierung wird vorausgesetzt)

Prinzipieller Algorithmus:

1. Es wird eine Variable L für die linke Grenze mit 1 und die rechte Grenze R mit dem größten Indexwert belegt.#
2. Der Feldinhalt an der Stelle (L+R)/2 wird mit dem gesuchten Wert verglichen
3. Falls der gesuchte Werte größer oder gleich ist, wird R = (L+R)/2 gesetzt, ansonsten wird L = (L+R)/2 gesetzt.
4. Falls L ungleich R ist, wird bei 2 fortgesetzt.

Aufwand: infolge der Halbierungen nur maximal **($\log_2 n$) Vergleiche**

Nachteile: Aufwand für das Sortieren (insbes. bei Einfügen neuer Elemente)

Suche nach Werten mittels Binärsuche

```
// Binaersuche
function binaersuche(a(100)as integer,n as integer,
  Suchwert as integer) as integer
  Dim links as integer, rechts as integer, mitte%
    links=1 :  rechts= n: mitte=0
     do
      mitte=(rechts-links)/2
      if  Suchwert> a(mitte)  then
          links=mitte+1
       else
          rechts=mitte-1;
      end if
     Rem Schleife bricht bereits bei Gleichheit ab
  loop while a(mitte)<>Suchwert and links> rechts
   binaersuche = mitte
  End function
  Suche liefert Treffer oder nächstgelegenen Wert !
```

Sortierverfahren

- Bei großen Datenmengen sind Sortierverfahren eine grundlegende Voraussetzung für ein effizientes und schnellen Suchen von Daten.
- Als Sortierung wird die Existenz einer Ordnungsrelation für alle Wertepaare innerhalb der Menge bezeichnet.
- Ordnungsrelationen können sich auf Vergleiche von Zahlen, Zeichen, Texten oder gemischten Daten beziehen. Die für eine Ordnungsrelation verwendeten Daten werden als Schlüssel bezeichnet. Es können für heterogene Daten auch mehrere Schlüssel definiert werden (z.b. Adressdaten / Telefondaten / Geburtstag)

Historisch haben sich verschiedene Verfahren herausgebildet.

- **Interne Verfahren :** arbeiten komplett im Hauptspeicher und haben alle Daten im Zugriff
- **Externe Verfahren :** müssen ganz oder teilweise auf Daten in Massenspeichern zugreifen und sind vor allen auf eine minimale Anzahl von externen Datenzugriffen optimiert (1 Mio. mal langsamer als Speichertransaktionen) – Hauptanwendung in Datenbanken
- Bei der Bewertung der Verfahren sind neben der reinen Zeitbetrachtung auch Anforderungen an den Speicherplatz und das Verhalten bei Veränderungen der Datenbasis von Bedeutung !

Einfache Sortierverfahren mit Wertetausch

Bubblesort (analog zu aufsteigenden Luftblasen) :

- Ein unsortiertes Feld wird durchlaufen.
  - Dabei werden benachbarte Feldinhalte miteinander verglichen.
  - Falls eine falsche Reihenfolge vorliegt, werden beide Werte vertauscht.
- Der Durchlauf wird N-1 mal oder solange Vertauschungen auftreten, wiederholt.
- Aufwand wächst quadratisch mit N an ($O(n^2)$)

```
Function bubblesort(a(100) as integer )
{ Dim i as integer, j as integer ,z as integer , flag as integer
 for  i = 0 to n
  flag = false
   for  j = 1 to  n – i
     if a(j-1) > a(j) then z= a(j-1):a(j-1) = a(j):a( j ) =z: flag = true: Rem Tausch
   next j
  if ( flag == false )   break; ‘ Ausstieg (falls fertig sortiert)  aus 1. Schleife
 next i
```

Quicksort

Quicksort ist eines der schnellsten Sortierverfahren.

- es ist ein Vertreter der sogenannte "Teile und Herrsche"-Algorithmen
- Problem wird in immer kleinere, analoge Teilprobleme zerlegt
- Ab einer bestimmten Stufe wird die Lösung des Teilproblems trivial oder mit einfacheren Algorithmen durchgeführt.

Algorithmus von Quicksort :

- Man teilt ein unsortiertes Feld anhand eines Schlüsselwertes in zwei Hälften.
- Durch paarweises Austauschen von der linken und rechten Seite her werden alle Werte kleiner als der Schlüsselwert in die linke Hälfte und alle Werte größer als der Schlüsselwert in die rechte Hälfte einsortiert.
- Im ungünstigsten Fall kann eine Seite dabei auch leer bleiben (der Schlüssel sollte daher vom Wert etwa in der Mitte liegen.)
- Für jede der beiden Hälften ruft man den Quicksort-Algorithmus rekursiv erneut auf.

Vorteil: sehr schnell für normal unsortierte Felder – îm Mittel $O(n \log_2 n)$

- Bei genau invertierter Reihenfolge ist Quicksort leider nur genauso so schnell wie Bubblesort.

Quicksort als C-Funktion

```
void quickSort( int a[], int links, int rechts )
{   int        i = links, j = rechts;  int       x, w;
    x = a[int((links+rechts)/2)];  // Anfangszerlegung
   do  {    while( a[i] < x ) i++;   // wandere solange ok nach rechts
            while( a[j] > x ) j--;    // wandere solange ok nach links
       if( i <= j )   {    w = a[i];   a[i] = a[j];  a[j] = w;  // Tausch
                           i++;  j--;  }
   } while( i <= j );
   if( links < j ) quickSort( a, links, j );        // Rekursiver Aufruf für linken Teil
   if( i < rechts ) quickSort( a, i, rechts );    // Rekursiver Aufruf für rechten Teil
}
```

VB-Version unter https://msdn.microsoft.com/de-de/library/bb979305.aspx

Vorteil: sehr schnell für normal unsortierte Felder – îm Mittel $O(n \log_2 n)$

- Bei genau invertierter Reihenfolge ist Quicksort leider nur genauso so schnell wie Bubblesort.

Sortieren durch Einfügen

Falls Daten bereits teilweise vorsortiert sind oder nur neue Elemente eingefügt werden müssen, ist nachfolgendes Verfahren von Vorteil:

Algorithmus von Sortieren durch Einfügen :

1. Man geht von einem bereits teilweise sortierten Feld aus. (Allgemein kann man auch nur das erste Element als sortiert auffassen -> Start dann bei 2).
2. Das nächste Element neben dem sortierten Teil wird in eine Zwischenvariable gespeichert und die Position als frei vermerkt.
3. Falls das Element links neben der aktuellen Position die Reihenfolgebedingung im Vergleich zur Zwischenvariable nicht erfüllt (z.B. größer als einzusortierendes Element) dann wird dieses Element um eine Position nach rechts verschoben, andernfalls wird das einzufügende Element an die aktuelle Position eingetragen.
4. Schritt ist zu wiederholen, solange nicht eingefügt werden konnte oder links noch Elemente existieren.

VB-Funktion für Sortieren durch Einfügen

```
Function InsertionSort(a() As Integer, n As Integer) As
    Integer
Dim i As Integer, j As Integer, flag As Integer
Dim elem As Integer: Rem aktives Data Element
For i = 2 To n
   elem = a(i): 'rem hole aktuelles Element
   j = i - 1:   flag = 0
 Do While elem < a(j) And flag = 0
        Rem Vergleiche Elemente
       a(j + 1) = a(j): ' Verschiebe nach oben
      j = j - 1: Rem Runterzählen
      If j < 1 Then flag = 1
  Loop
  a(j + 1) = elem: Rem Einfügen des neu sortierten El.
  flag = 0
Next i
End Function
```

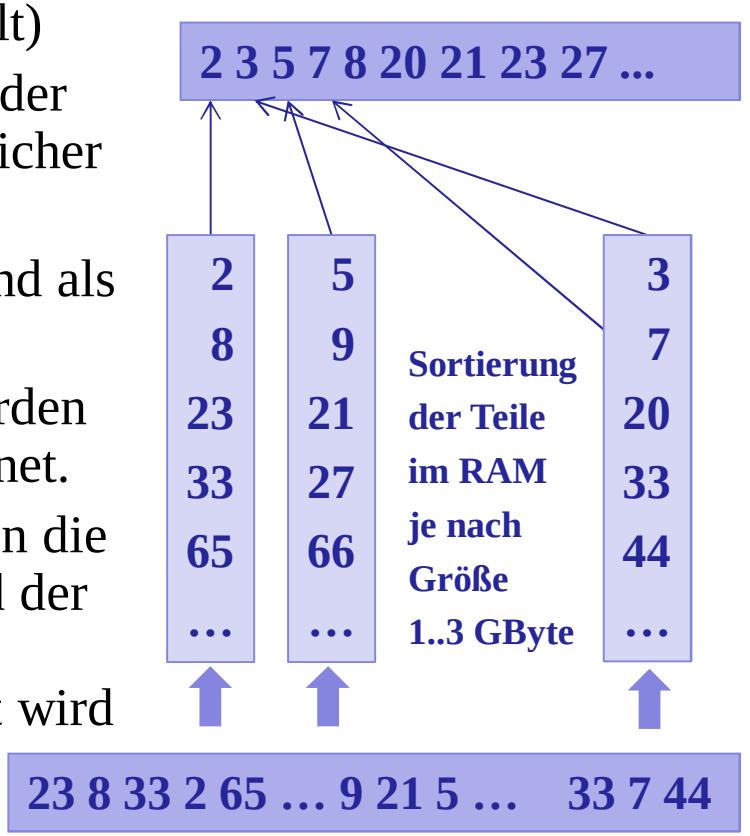
Sortieren sehr großer Datenmengen

- sehr große Datenmengen konnten insbesondere in der Vergangenheit nicht im Hauptspeicher sortiert werden - z.B. sehr große Meßdatenbestände aus Astronomie, Kernphysik, Gentechnik (mehrere Gigabyte bis Terrabyte)
- Ausweg: Sortierung unter Nutzung **externer Massenspeicher** (Festplatten etc.)

Prinzipielle Vorgehensweise

(die Dateifunktionen werden im 2. Semester behandelt)

1. Die Daten werden in N Segmente entsprechend der maximal möglichen Verarbeitungsgröße im Speicher aufgeteilt (z.B. 64 Mbyte … 2..3 GByte …)
2. Jedes Datensegment wird im Speicher sortiert und als einzelne Datei wieder abgelegt.
3. Nach dem internen Sortieren aller Segmente werden alle N Dateien gleichzeitig im Lesemodus geöffnet.
4. Durch paralleles Einlesen aller N Dateien werden die jeweils ersten Werte miteinander verglichen und der kleinste/bzw. größte in eine Ergebnisdatei geschrieben. Aus dem entsprechendem Segment wird danach der nächste Wert nachgeladen.
5. Solange noch Werte vorhanden sind, wird 4. wiederholt.



Zusammenfassung zum Thema Felder

- Felder (engl. Arrays) sind in der Informatik das wichtigste Ablagemittel für Daten
- Mit Schleifen über Feldern können sehr effiziente Verarbeitungsalgorithmen entwickelt werden.

- Viele Such- und Verarbeitungsalgorithmen arbeiten besonders schnell auf sortierten Feldern.
- Sortieralgorithmen bilden daher den Schwerpunkt bei der Arbeit mit Feldern in der Informatik.
- Weitere, spezielle Sortierverfahren sind in Datenbanken und speziellen Tools verfügbar.