Kapitel 7

Zusammengesetzte Datentypen, Vektoren, Zeichenketten

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Kapitel 7 – Zusammengesetzte Datentypen

7.1. Vektoren

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Es soll ein Programm zur Versuchsauswertung erstellt werden:

- Bis zu 100 Messwerte einlesen
- Messwerte zur Kontrolle wieder ausgeben
- Mittelwert und Maximum berechnen

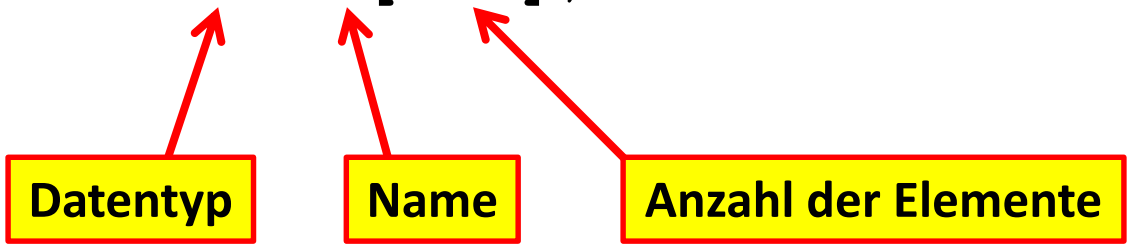
Es werden dazu 100 Variablen vom Typ float oder double benötigt. Es wäre möglich – aber sehr umständlich – zu diesem Zweck 100 verschiedene Variablen zu definieren.

(Was würde bei einer Erweiterung auf 1000 Werte passieren?)

Einfacher: Definition eines Vektors mit 100 Elementen

Definition eines Vektors:

```
double x[100];
```



- Reserviert 100 Speicherplätze vom Typ double
- Die Anzahl der Elemente sollte **als Integer-Konstante** angegeben werden, Folgendes ist in älteren Varianten von C sowie in C++ **nicht erlaubt**: `float x[dim];`

  Fehler: „dim“ ist keine Konstante
- Die einzelnen Elemente des Vektors liegen im Hauptspeicher in aufeinanderfolgenden Speicherzellen
- ```
  float x[4] = {2.0, 4.0, 0.0, 1.2};
  int z[3] = {1, 5, 10};
  int z[] = {1, 5, 10};
  ```

  Definition und Initialisierung zugleich!
  ```

Zugriff auf einzelne Elemente:

`x[0] = 5.5;`	Zuweisung an Element Nr. 0
`x[1] = y;`	Zuweisung an Element Nr. 1
`x[k] = y;`	(k+1)-tes Element des Vektors, k muss Integervariable oder -konstante sein

Ein einzelnes Element x[k] des Vektors kann wie eine normale double-Variable verwendet werden:

```
x[k] = x[3];
printf("Sechster Wert im Vektor: %f", x[5]);
if (x[i] > x[i+1])
    printf("Groesser!\n");
```

Beachte:

- Index läuft von 0 bis 99 (und nicht etwa von 1 bis 100)
- Index darf nicht zu groß werden; x[391] wird vom Compiler nicht erkannt!

Teilaufgaben/Funktionen:

1) `int einlesen(void)`
   - Zunächst Anzahl der vorhandenen Messwerte abfragen
   - Messwerte der Reihe nach einlesen, Elemente x[0] bis x[zahl-1] belegen
   - Aufrufparameter: keine (void)
   - Rückgabewert: Anzahl (int)
2) `void ausgeben(int n)`
   - Aufrufparameter: n – Zahl der eingelesenen Messwerte
   - *Funktion muss wissen, wie viele Elemente von x tatsächlich Messwerte sind*
   - Rückgabewert: keiner (void)
3) `double mittelwert(int n)`
   - Aufrufparameter: n – Zahl der eingelesenen Messwerte
   - Rückgabewert: Mittelwert (double)
4) `double maximum(int n)`
   - Aufrufparameter: n – Zahl der eingelesenen Messwerte
   - Rückgabewert: Maximum (double)

```c
#include <stdio.h>
#define SIZE 100

int einlesen(void);
void ausgeben(int n);
double mittelwert(int n);
double maximum(int n);

double x[SIZE];

int main(void)
{
   double xmit, xmax; int anzahl;
   anzahl = einlesen();          /* Messwerte eingeben   */
   ausgeben(anzahl);             /* Messwerte ausgeben   */
   xmit = mittelwert(anzahl);    /* Mittelwert berechnen */
   xmax = maximum(anzahl);       /* Maximum berechnen    */

   printf("Mittelwert = %f\n", xmit);
   printf("Maximalwert = %f\n", xmax);
   return 0;
}
```

Hauptprogramm, Funktion `main`

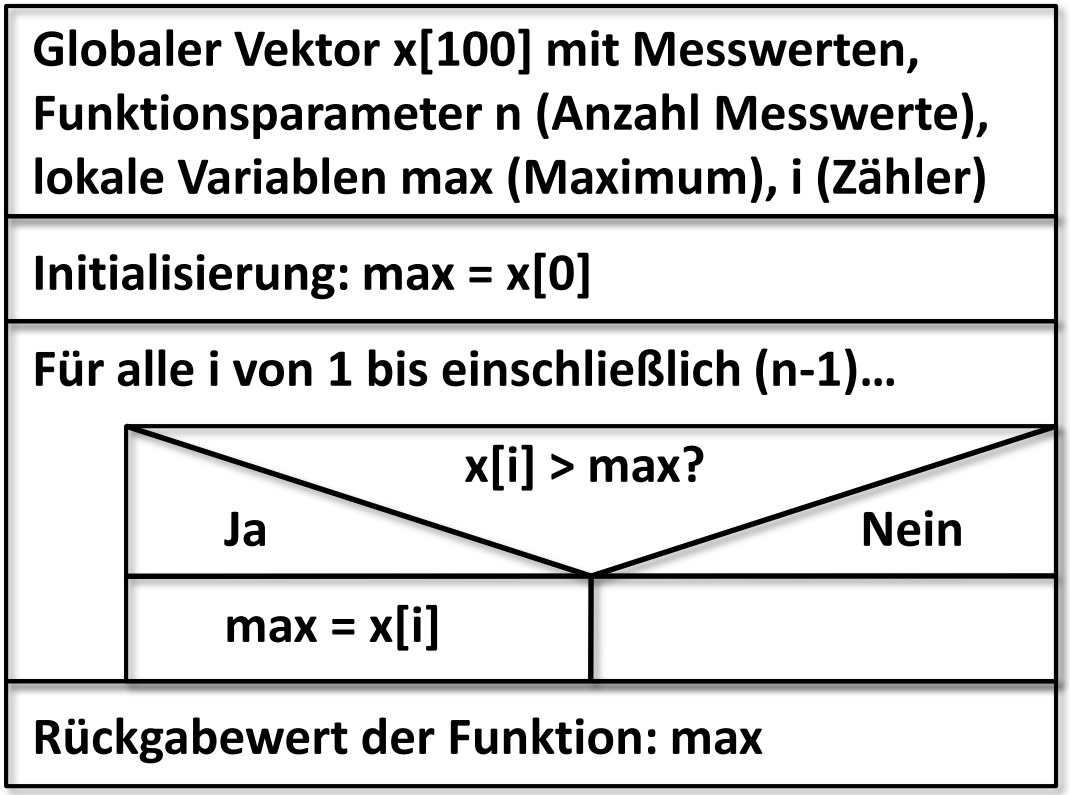
Definition eines Vektors mit 100 double-Elementen

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Beispiel: Funktion **maximum** zur Ermittlung des max. Messwerts.

```
/* Es werden alle n Messwerte in dem Vektor x[]
   durchsucht; das Maximum wird zurückgegeben */
double maximum(int n)
{
  double max;
  int i;
  max = x[0];
  for(i=1; i<n; i++)
  {
    if(x[i] > max)
      max = x[i];
  }
  return max;
}
```

Was passiert bei n == 0…?



Aufgabe:

- Erstellen Sie die Struktogramme und C-Quelltexte der fehlenden Funktionen **`einlesen`**, **`ausgeben`** und **`mittelwert`**!
- Übersetzen Sie das Gesamtprogramm und testen Sie es!
- Was könnte noch verbessert werden? (Zum Beispiel die Eingabe der Messwerte: Nicht erst die Anzahl eingeben und danach alle Messwerte, sondern direkt alle Messwerte eingeben und durch Eingabe von -1 beenden. Das Programm kann dann die Anzahl automatisch ermitteln…)
- Geben Sie die Messwerte nicht über die Tastatur ein, sondern speichern Sie die Werte zunächst in einer Textdatei. Rufen Sie dann Ihr Programm über die DOS-Eingabeaufforderung auf und lesen Sie die Messwerte per Eingabeumlenkung ein!

Kapitel 7 – Zusammengesetzte Datentypen

7.2. Sortieren eines Vektors

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Aufgabe:

Erweitern Sie das Messwertprogramm wie folgt:

Nach dem Einlesen der Messwerte sollen diese zunächst sortiert und erst danach ausgegeben werden!

- Schreiben Sie dazu eine Funktion **`void sortiere(int n)`**
- Die Elemente von x müssen so umgeordnet werden, dass gilt:

  x[0] ≤ x[1] ≤ x[2] ≤ ... ≤ x[n-1] (für alle n Elemente)
- Verwenden Sie zum Sortieren keinen zweiten (Hilfs-)Vektor, sondern führen Sie die Sortierung direkt auf den Elementen des Vektors x[] durch!

Idee: Sortierung des Vektors mittels „Bubble-Sort“

Erster Durchlauf:

(**5 1** 4 2 8) → (**1 5** 4 2 8) Die ersten zwei Elemente vergleichen, tauschen
(1 **5 4** 2 8) → (1 **4 5** 2 8)
(1 4 **5 2** 8) → (1 4 **2 5** 8)
(1 4 2 **5 8**) → (1 4 2 **5 8**) Reihenfolge stimmt schon, nicht tauschen

Zweiter Durchlauf:

(**1 4** 2 5 8) → (**1 4** 2 5 8)
(1 **4 2** 5 8) → (1 **2 4** 5 8) Elemente tauschen
(1 2 **4 5** 8) → (1 2 **4 5** 8)
(1 2 4 **5 8**) → (1 2 4 **5 8**)

Die rot markierten Elemente werden verglichen und ggf. vertauscht

Dritter Durchlauf:

(**1 2** 4 5 8) → (**1 2** 4 5 8) Im dritten Durchlauf werden keine Elemente
(1 **2 4** 5 8) → (1 **2 4** 5 8) mehr vertauscht; der Algorithmus erkennt, dass
(1 2 **4 5** 8) → (1 2 **4 5** 8) der Vektor jetzt sortiert ist und stoppt
(1 2 4 **5 8**) → (1 2 4 **5 8**)

7.2. Sortieren eines Vektors


```
void sortiere(int n) /* x[] mittels Bubble-Sort sortieren */
{
    int i; double hilf; char sortiert;

    do   /* Schleife wiederholen, bis Vektor sortiert ist */
    {
        sortiert = 'j';       /* Annahme, x[] ist sortiert */
        for(i = 1; i < n; i++)
        {
            if(x[i] < x[i-1])
            {
                hilf = x[i];      /* x[i], x[i-1] tauschen */
                x[i] = x[i-1];
                x[i-1] = hilf;
                sortiert = 'n';  /* ...war nicht sortiert */
            }
        }
    }
    while(sortiert == 'n');
}
```

Implementierung des Bubble-Sort-Algorithmus

Kapitel 7 – Zusammengesetzte Datentypen

Problem:

Es soll ein C-Programm zum Rechnen mit (2-, 3-, mehrdimensionalen) Matrizen erstellt werden. Bislang sind aber nur eindimensionale Vektoren bekannt.

Anwendung von Matrizen: Lineare Gleichungssysteme, Finite-Elemente-Methode (sehr große Matrizen!), Drehimpuls/Trägheitsmatrix, Mathematik/ Geometrie/Computergrafik usw.

$$\begin{aligned} 7 \cdot x_1 + 8 \cdot x_2 &= 17 \\ 2 \cdot x_1 + 4 \cdot x_2 &= 11 \end{aligned} \quad \rightarrow \quad \textit{(als Matrix)} \quad \begin{bmatrix} 7 & 8 \\ 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 17 \\ 11 \end{bmatrix}$$

Eine Matrix besitzt Zeilen und Spalten → im C-Programm wird eine Datenstruktur für zwei-/mehrdimensionale Felder benötigt.

Definition eines zweidimensionalen Feldes:

```
double a[3][3];
```

Zugriff auf einzelne Elemente:

```
a[0][0] = 5.5;
a[0][1] = 1.5;
a[1][0] = 0.5;
a[i][k] = 1.7;
```

mit $0 \leq i < 3$ und $0 \leq k < 3$

(i und k sind Variablen vom Typ Integer)

7.3. Mehrdimensionale Felder

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN · FH MÜNCHEN

Elemente in Matrixform:

$$a = \begin{vmatrix} a[0][0] & a[0][1] & a[0][2] \\ a[1][0] & a[1][1] & a[1][2] \\ a[2][0] & a[2][1] & a[2][2] \end{vmatrix} = \begin{vmatrix} 5.5 & 1.5 & .. \\ 0.5 & .. & .. \\ .. & .. & .. \end{vmatrix}$$

Spalten

Zeilen

Anordnung im Hauptspeicher:

a[0][0]	5.5
a[0][1]	1.5
a[0][2]	
a[1][0]	0.5
a[1][1]	
...	

Die Matrix wird zeilenweise im Hauptspeicher abgelegt!

Beispiele:

1) `double b[2][3];` (nicht quadratisch – 2 Zeilen, 3 Spalten)

2) Definition und gleichzeitige Initialisierung:
```
double b[2][3] = { {2.0,4.0,0.0}, {2.2,4.2,1.2} };
```

3)
```
#define   ZEILEN   1000
#define   SPALTEN   500
double d[ZEILEN][SPALTEN];
```

Zeilen- und Spaltenanzahl sollten mit Konstanten angegeben werden!

4) `double temp[10][10][10];`
3-dimensionales Feld – zum Beispiel Temperaturfeld im Raum:
Jedem Gitterpunkt im Raum wird eine Temperatur zugeordnet

5) `double v[100][100][200][3];` (4 dimensionales Feld)
Jedem Gitterpunkt im Raum werden 3 Werte zugeordnet –
z. B. drei Geschwindigkeitskomponenten in einem Strömungsfeld:

`v[2][4][7][0]` v_x am Punkt x=2, y=4, z=7
`v[2][4][7][1]` v_y am Punkt x=2, y=4, z=7

$100 \times 100 \times 200 \times 3 = 6 \times 10^6$ Elemente
Speicherplatzbedarf $6 \times 10^6 \times 8$ Byte = 48 MB

Kapitel 7 – Zusammengesetzte Datentypen

7.4. Umgang mit ein-/zweidimensionalen Feldern

Beispiel 1: Alle Elemente der Matrix a auf 0 setzen

Beispiel 2: Zwei Matrizen addieren: c = a + b

Beispiel 3: Prüfen, ob Matrix a symmetrisch ist

(symmetrische Matrizen sind „gutartig“, sie haben reelle Eigenwerte und N linear unabhängige Eigenvektoren…)

Beispiel 4: Matrix mit Vektor multiplizieren: $y = a \times x$

Beispiel 5: Multiplikation von zwei Matrizen: $c = a \times b$

```
#define N 10
double a[N][N], b[N][N], c[N][N];
double x[N], y[N];
```

Kapitel 7 – Zusammengesetzte Datentypen

Frage:

Wie werden Zeichenketten („Strings“) in der Programmiersprache C abgespeichert und verarbeitet?

Erster Ansatz: char-Vektor verwenden

```
char wort[5] = { 'H','a','l','l','o' };

int i;
for(i = 0; i < 5; ++i)
     printf("%c", wort[i]);
```

Nachteile:

- Initialisierung, Ausgabe ist umständlich – viel Schreibarbeit
- Bei der Bearbeitung (z. B. Ausgabe mittels printf), muss man sich merken, wie viele Buchstaben das Wort besitzt.

7.5. Zeichenketten, Strings

Wörter werden als Zeichenketten („Strings“) abgespeichert. Strings besitzen einige Besonderheiten bezüglich Initialisierung, Länge und Endekennung, sind aber im Wesentlichen normale char-Vektoren:

```
char s[]="Hallo";
```

- Erzeugt und initialisiert den char-Vektor s
- Binäre Null als Endekennung, binäre Null ≠ ASCII-Zeichen '0'!
- Anzahl Elemente: Buchstabenzahl + 1
- Länge des Vektors wird vom Compiler berechnet – über rechte Seite bestimmt
- Binäre Null wird als Endekennung angehängt

s[0]	H	72	0100 1000
s[1]	a	97	0110 0001
s[2]	l	108	0110 1100
s[3]	l	108	0110 1100
s[4]	o	111	0110 1111
s[5]	\0	0	0000 0000

Ein String ist ein char-Vektor mit einer binären Null zur Endekennung

7.5. Zeichenketten, Strings

Beispiel 1: Zeichen 'o' in "Hallo" durch 'e' ersetzen (→ "Halle")

Beispiel 2: Ausgabe eines Strings mittels printf

Beispiel 3: Einlesen eines Strings von der Tastatur

Beispiel 4: Klein- in Großbuchstaben umwandeln

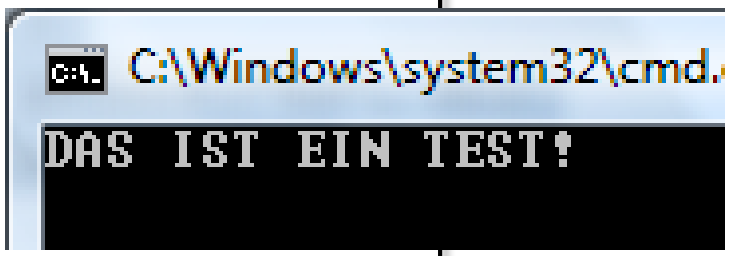
Beispiel 5: Headerdatei string.h mit vielen Funktionen zur Arbeit mit Zeichenketten

Beispiel 4: Klein- in Großbuchstaben umwandeln:

```
#include <stdio.h>
#include <ctype.h>

int main(void)
{
    int i = 0;
    int diff = 'a' - 'A';

    char s[] = "Das ist ein Test!";
    while(s[i] != 0)
    {
        if('a'<=s[i] && s[i]<='z')
            s[i] = s[i] - diff;
        ++i;
    }
    printf("%s", s);
    return 0;
}
```



Beispiel 5: Headerdatei string.h

- `int strlen(str);`
  - Gibt die Länge eines Strings zurück, zählt Zeichen bis zur Endekennung

    `i = strlen("Hallo");` gibt 5 zurück

    (Mit binärer Null benötigt der String 6 Elemente im char-Vektor!)
- `int strcmp(str1, str2);`
  - Gibt 0 zurück, wenn beide Zeichenketten gleich sind
  - Ergebnis ist > 0, wenn str1 lexikalisch größer ist als str2
  - Ergebnis ist < 0, wenn str1 lexikalisch kleiner ist als str2

    `"z" > "a"  "ab" < "abc"  "ad" < "af"  "ac" > "abc"`
- **Viele weitere Funktionen**
  - Kopieren, Anhängen, Unterteilen von Zeichenketten, Suchen in Zeichenketten usw.
  - Vergl. Online-Hilfe zu string.h!