

PROGRAMMIEREN I

WS 2022

Prof. Dr.-Ing. Kolja Eger
Hochschule für Angewandte Wissenschaften Hamburg

Check-In



Präsenzklausur im Labor (180min)

Dynamische Speicherallokation

Praktikum 7

Strukturen

Praktikum 6

Dateien

Praktikum 5

Zeiger

Praktikum 4

Vektoren (Arrays)

Praktikum 3

Funktionen

Praktikum 2

Kontrollstrukturen

Datentypen & Operatoren

Erste Programme

Praktikum 1

Unser Weg durch das Semester

VEKTOREN (ARRAYS)

Eindimensionale Vektoren

- **Allgemein** lautet die Syntax für Vektoren

```
<Datentyp> <Variablenname>[<Anzahl>]
```

Beispiel: Um einen Notenspiegel zu berechnen, definieren Sie einen Vektor mit der Anzahl der Einsen, Anzahl der Zweien, .. und der Sechsen

```
int Noten[6];
```

Vektoren – Beispiel: Notenspiegel

```
int Noten[6];
```

- Im Arbeitsspeicher wird Platz für sechs int-Variablen geschaffen (allokiert)
- Alle Variablenwerte sind noch undefiniert

Typ:	int	int	int	int	int	int
Name:	Note[0]	Note[1]	Note[2]	Note[3]	Note[4]	Note[5]
Speicher:						
Wert:	?	?	?	?	?	?

Initialisierung eines Vektors – Weg 1: Nach der Definition

- Zugriff auf Elemente mit eckigen Klammern und Index
- Achtung: Index startet mit null und endet eins vor der Anzahl der Elemente
- Beispiel:

```
// Initialisierung nach Definition  
Noten[0] = 0  
Noten[1] = 0  
Noten[2] = 0;  
Noten[3] = 0;  
Noten[4] = 0;  
Noten[5] = 0;
```

Initialisierung eines Vektors – Weg 2: Direkt bei der Definition

- Werte können in geschweiften Klammern bei der Definition angegeben werden
- Elemente werden durch Komma getrennt
- Anzahl der Elemente muss nicht in eckigen Klammern angegeben werden, sondern kann über die Anzahl der Elemente in geschweiften Klammern abgeleitet werden

- Beispiel:

```
int Noten_v2[] = { 0,0,0,0,0,0 };
```

- Falls Anzahl der Elemente in eckigen Klammern angegeben ist und die Initialisierung nicht alle Elemente enthält, werden alle anderen Elemente auf null gesetzt

- Beispiel:

```
int Noten_v3[6] = { 0 };
```


Arbeiten mit Vektoren

- Zugriff auf Elemente mit eckigen Klammern und Index
- Achten Sie auf die Grenzen des Arrays!
- Beispiele:

```
Noten[2]++;
```

```
int Bestanden = Noten[0] + Noten[1] + Noten[2] + Noten[3];
```

```
int Zahl = 3;  
if (Zahl >= 1 && Zahl <= 6)  
    Noten[Zahl - 1]++;
```

Vektoren als Funktionsparameter (II)

- Beispiel:

```
//void Init(int Note[6]); // Vektor als Funktionsparameter
void Init(int Note[]); // Anzahl der Elemente kann weggelassen werden

int main()
{
    int Note_Sem_1[6];

    Init(Note_Sem_1);

    return 0;
}

//void Init(int Note[6])
void Init(int Note[]) // Anzahl der Elemente kann weggelassen werden
{
    int i;

    for (i = 0; i < 6; i++) Note[i] = 0;
}
```

Übung

- Bei uns werden Notenpunkte vergeben (siehe rechts)
- Erstellen Sie ein Array für die Notenpunkte
- Berechnen Sie den Notendurchschnitt für folgende Verteilung und geben sie ihn aus

1x 15 Punkte	4x 8 Punkte
3x 14 Punkte	3x 7 Punkte
1x 13 Punkte	4x 6 Punkte
0x 12 Punkte	2x 5 Punkte
7x 11 Punkte	1x 4 Punkte
3x 10 Punkte	1x 3 Punkte
5x 9 Punkte	1x 0 Punkte

Zusatzaufgabe: Geben Sie zu dem Notendurchschnitt auch die Benotung in Worten aus (z.B. „gut“).

Notenpunkte	Dezimalzahlen- bewertung		Note (Benotung)
15	0.7	=	ausgezeichnet
14 und 13	1.0 und 1.3	=	sehr gut
12, 11 und 10	1.7, 2.0 und 2.3	=	gut
9, 8 und 7	2.7, 3.0 und 3.3	=	befriedigend
6 und 5	3.7 und 4.0	=	ausreichend
4 bis 0	4.3 bis 5.0		nicht
4	4.3		ausreichend
3	4.7		
2 bis 0	5.0		

Übungsaufgaben*

- Definieren und initialisieren Sie folgende Vektoren:
 - a) Einen Vektor, der die 12 Monatsumsätze eines Artikels speichern kann. Die ersten drei Elemente werden mit den schon bekannten Umsätzen 876,54 und 789,12 und 657,45 initialisiert
 - b) Einen Vektor mit 8 ganzzahligen Elementen, der mit den Potenzen 2^0 , 2^1 , 2^2 , ..., 2^7 , also mit 1, 2, 4, ..., 128 initialisiert ist.



(*) aus „C Das Übungsbuch“ von Peter Prinz

Mehrdimensionale Vektoren

- Auch Vektoren lassen sich verketten
- Aus eindimensionalen Vektoren werden zwei- oder mehrdimensionale Vektoren
- **Anzahl der Zahlen in eckigen Klammern** gibt die **Dimension** an
- Syntax für **zweidimensionale Vektoren**

<Datentyp> <Variablenname>[<Anzahl>][<Anzahl>]

- und dreidimensionale Vektoren (usw.)

<Datentyp> <Variablenname>[<Anzahl>][<Anzahl>][<Anzahl>]

Mehrdimensionale Vektoren - Beispiel

- Umrechnungstabelle für Fahrenheit/Celsius



Fahrenheit	Celsius
0	-17,78
50	10
100	37,78
150	65,56

- Vektor definieren

```
float FtoC[4][2];
```

Speicherabbild für

float FtoC[4][2];

Typ:	float	float	float	float	float	float	float	float
Name:	FtoC[0][0]	FtoC[0][1]	FtoC[1][0]	FtoC[1][1]	FtoC[2][0]	FtoC[2][1]	FtoC[3][0]	FtoC[3][1]
Speicher:								
Wert:	?	?	?	?	?	?	?	?

- Speicher ist eindimensional organisiert
- Mehrdimensionale Vektoren werden sequentiell (d.h. hintereinander) gespeichert

→ Beispiel in Visual Studio

```
#include <stdio.h>

void InitTemperatur(float FtoC[][2]);

int main(void) {

    // 1.Weg: Initialisierung bei Definition
    float FtoC[4][2] = { {0, -17.78f},
                        {50, 10},
                        {100, 37.78f},
                        {150, 65.56f} };

    // 2.Weg: Init von Einzelwerten nach Definition ..
    FtoC[0][0] = 0;
    FtoC[0][1] = -17.78f;
    FtoC[1][0] = 50;
    // ..

    // .. oder innerhalb einer Schleife
    for (int i = 0; i < 4; i++) {
        FtoC[i][0] = (50.0f * i);
        FtoC[i][1] = (5.0f / 9.0f * (FtoC[i][0] - 32.0f));
    }

    // Arbeiten mit mehrdimensionalen Vektoren
    for (int i = 0; i < 4; i++)
        printf("%6.2f F -> %6.2f C\n", FtoC[i][0], FtoC[i][1]);

    // Funktionsaufruf mit 2D-Array als Parameter
    InitTemperatur(FtoC);

    return 0;
}
```

```
void InitTemperatur(float FtoC[][2])
{
    for (int i = 0; i < 4; i++) {
        FtoC[i][0] = 50.0 * i; // 50er Schritte für
                               // Fahrenheit-Werte
        FtoC[i][1] = 5.0 / 9.0 * (FtoC[i][0] - 32.0);
        // Umrechnung in Celsius
    }
}
```

Initialisierung von mehrdimensionale Vektoren

- Auch hier kann eine Initialisierung direkt bei der Definition erfolgen
- Verkettung mit geschweiften Klammern

```
float FtoC[4][2] = { {0, -17.78f},  
                     {50, 10},  
                     {100, 37.78f},  
                     {150, 65.56f} };
```

- Warum steht hinter einigen Zahlen ein *f*?
 - Fließkommazahlen werden als *double* interpretiert und eine Warnung wird ausgegeben, dass bei der Umwandlung in ein *float* der Wert verkürzt wird
 - Mit *f* wird angegeben, dass die Konstante vom Datentyp *float* ist

Initialisierung von mehrdimensionale Vektoren nach der Definition

- Alternativ kann der Vektor auch wieder nach der Definition initialisiert werden

Option 1: als Einzelwerte


```
// Init von Einzelwerten  
FtoC[0][0] = 0;  
FtoC[0][1] = -17.78f;  
FtoC[1][0] = 50;
```

Option 2: in einer Schleife

```
for (int i = 0; i < 4; i++) {  
    FtoC[i][0] = (50.0f * i);  
    FtoC[i][1] = (5.0f / 9.0f * (FtoC[i][0] - 32.0f));  
}
```


Initialisierung von mehrdimensionale Vektoren - Speicherabbild

- Nach der Initialisierung (egal welcher Weg gewählt wurde) sieht das Speicherabbild wie folgt aus

Typ:	float	float	float	float	float	float	float	float
Name:	FtoC[0][0]		FtoC[1][0]		FtoC[2][0]		FtoC[3][0]	
		FtoC[0][1]		FtoC[1][1]		FtoC[2][1]		FtoC[3][1]
Speicher:								
Wert:	0,0	-17,78	50,0	10,0	100,0	37,78	150,0	65,56

Arbeiten mit mehrdimensionale Vektoren

- Zugriff auf Elemente erfolgt durch eckige Klammern
- Abhängig von der Dimension werden mehrere eckige Klammern hintereinander angegeben
- Grenzen des Vektors müssen eingehalten werden → liegt in der Verantwortung der Programmiererin/des Programmierers

```
// Arbeiten mit mehrdimensionalen Vektoren
for (int i = 0; i < 4; i++)
    printf("%6.2f F -> %6.2f C\n", FtoC[i][0], FtoC[i][1]);
```

Ausgabe:

Microsoft Visual Studio-Debugging-Konsole

```
0.00 F -> -17.78 C
50.00 F -> 10.00 C
100.00 F -> 37.78 C
150.00 F -> 65.56 C
```

Mehrdimensionale Vektoren als Funktionsparameter

- Wie bei eindimensionalen Vektoren können auch mehrdimensionale Vektoren als Parameter einer Funktion übergeben werden
- Beispiel für eine Deklaration

```
void InitTemperatur(float FtoC[4][2]);
```

- Können wir die Vektorgröße auch hier wieder weglassen?
 - Es kann nur die erste Dimension weggelassen werden
 - Die anderen Dimensionen werden vom Compiler benötigt, um die Position im Speicher zu bestimmen

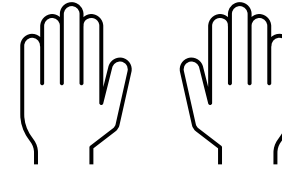
```
void InitTemperatur(float FtoC[][2]);
```

ZAHLENSYSTEME

Zahlensysteme - Überblick

- Ein Zahlensystem legt fest wie eine Zahl dargestellt wird
- **Dezimalsystem** ist unser gebräuchliches Zahlensystem und wir nutzen es alltäglich
- In der Computertechnik werden andere Zahlensysteme verwendet
 - **Dualsystem / Binärsystem**
 - Hexadezimalsystem
 - Oktalsystem

Dezimalsystem



- Dezimalsystem hat die Basis 10, das heißt
 - Für jede Ziffer stehen zehn unterschiedliche Symbole zur Verfügung
→ (0, 1, 2, 3, 4, 5, 6, 7, 8 und 9)

- Gewichtung der Stellen

$$\begin{aligned} 1234_{10} &= 1 * 10^3 + 2 * 10^2 + 3 * 10^1 + 4 * 10^0 \\ &= 1000 + 200 + 30 + 4 \end{aligned}$$

Beachte:
Die Basis kann als Suffix
(tiefgestellt) bei der Zahl
angegeben werden

Hier: 1234_{10}

- Wert einer Ziffer: Ziffer * Basis^{Stelle}

Dualsystem/Binärsystem

- Typischerweise haben Schaltkreise in einem Computer nur zwei Zustände (Ausnahme Analogrechner oder Quantencomputer)
 - Spannung oder Nichtspannung
 - Eins oder Null
 - Low oder High
- Diese Information wird auch als 1 Bit bezeichnet
- Die Basis im Binärsystem ist die 2
- Wert kann wieder mit Ziffer * Basis^Stelle berechnet werden
- Beispiel:

$$\begin{aligned} 1101_2 &= 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 \\ &= 8 + 4 + 0 + 1 = 13_{10} \end{aligned}$$

Übung

- Berechnen Sie den dezimal Wert von

111_2

10100_2

Bis 1023 zählen – Mit den Fingern! | DieMaus | WDR

https://www.youtube.com/watch?v=OkcVk_PGYL4

Hexadezimalsystem

- Binärzahlen können sehr lang werden
- Ein Integer-Wert in Visual Studio hat 32bit
 - Beispiel: $10011001100110011001100110011001_2$
 - $2^{32} = 4\,294\,967\,296$ unterschiedliche Zahlen darstellbar
- Mit Hexadezimalzahlen können binäre Zahlen kürzer und besser lesbar dargestellt werden
- Basis ist 16 (2^4)
 - 4 Ziffern einer Binärzahl können zusammengefasst werden
- Folgende Ziffern werden genutzt (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)

Hexadezimalsystem - Beispiel

- Mögliche Ziffern: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F)
- Beispiel:

$$\begin{aligned} 1A0F_{16} &= 1 * 16^3 + 10(A) * 16^2 + 0 * 16^1 + 15(F) * 16^0 \\ &= 1 * 4096 + 10 * 256 + 0 * 16 + 15 * 1 \\ &= 6671_{10} \end{aligned}$$

- Übung

AFFE₁₆

Oktalzahlen

- Basis 8 (2^3)
 - 3 Stellen einer Binärzahl zusammenfassen
- Mögliche Ziffern: (0, 1, 2, 3, 4, 5, 6, 7)
- Übung: Welche Dezimalzahl steckt hinter 1234_8 ?

Umrechnung ins Dezimalsystem (allgemein)

- Zu konvergierende Zahl: $\dots z_3 z_2 z_1 z_0, z_{-1} z_{-2} z_{-3} \dots$
 - Basis der Zahl ist B
 - Dezimalzahl a ergibt sich mit $a = \sum_i z_i \cdot B^i$
- ➔ Um eine Zahl eines beliebigen Zahlensystems dezimal darzustellen, addiere man das Produkt jeder Ziffer mit der jeweiligen Gewichtung der Ziffer
- Nachkommastellen haben negative Gewichtung, z.B.

$$\begin{aligned} 0,1101_2 &= 1 * 2^{-1} + 1 * 2^{-2} + 0 * 2^{-3} + 1 * 2^{-4} \\ &= 0,5 + 0,25 + 0 + 0,0625 = 0,8125_{10} \end{aligned}$$

VIELEN DANK FÜR IHRE AUFMERKSAMKEIT!