

Datentypen: Enum, Array, Struct, Union

C-Kurs 2013, 2. Tutorium

Freitagsrunde

<http://wiki.freitagsrunde.org>

10. September 2013



This work is licensed under the *Creative Commons Attribution-ShareAlike 3.0 License*.

- 1 enum
 - Boolean als Enum
 - Übungsaufgabe: colors mit enum
- 2 Arrays
 - Java-Arrays vs. C-Arrays
- 3 struct
- 4 union
- 5 Zusammenfassung: Datenstrukturen
- 6 Zusatzaufgabe: Arrays wie in Java



4!

Aufzählungen(Enums)

```
1 int
```

```
1 double
```

```
1 enum day_t {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
```

- ▶ Neu definierter Datentyp: **enum day_t** mit 7 möglichen Werten
- ▶ Der Compiler wandert die Elemente des *enum* in Integer um

Enum

```
1 enum day_t {Mon, Tue, Wed, Thu, Fri, Sat, Sun};
2 enum day_t current_day = Tue;
3
4 switch (current_day) {
5 case Sat:
6 case Sun:
7     printf("Es ist Wochenende! \n");
8     break;
9 default:
10    printf("Es ist unter der Woche. \n");
11    break;
12 }
```

- ▶ werden auch als Aufzählungstypen bezeichnet
- ▶ Intern als Integer gehandelt
- ▶ Elemente sind nacheinander durchnummeriert
- ▶ Beispielsweise lassen sich die Wochentage gut repräsentieren
- ▶ Wie wäre es mit Boolean?



4!

Boolean repräsentiert durch Enums

- ▶ In C entspricht **false** dem Wert 0
- ▶ alles ungleich 0 ist **true**

```
1 enum bool {FALSE, TRUE};
```

Wenn man nichts angibt, wird das erste Element 0, das zweite 1, usw
Man kann auch diese Konstanten direkt einen Wert explizit zuweisen:

```
1 enum bool {  
2 TRUE = 42,  
3 FALSE = 0  
4 };
```

Enum-Boolean mit typedef

- ▶ Verwendet kann es werden mit

```
1 enum bool{FALSE,TRUE};  
2  
3 enum bool flag = TRUE;
```

- ▶ Sieht jedoch ungewöhnlich aus
- ▶ Besser ist die Verwendung mit **typedef**

```
1 typedef enum bool{FALSE, TRUE} boolean;  
2  
3 boolean flag = TRUE;
```

Colors mit enum

Schreibe eine Datenstruktur `Color`, die die Grundfarben **black**, **white**, **red**, **yellow**, **green**, **cyan**, **blue**, und **magenta** definiert, indem den einzelnen Varianten explizit HEX-Werte zugewiesen werden!

```
1 typedef enum color{  
2     ...  
3     ...  
4 } Color;
```

Lösung der Übungsaufgabe

```
1 typedef enum color{
2     WHITE= 0xFFFFFFFF ,
3     BLACK= 0x000000 ,
4     RED= 0xFF0000 ,
5     YELLOW= 0xFFFF00 ,
6     GREEN= 0x00FF00 ,
7     CYAN= 0x00FFFF ,
8     BLUE= 0x0000FF ,
9     MAGENTA= 0xFF00FF
10 } Color;
11
12 //Verwendung:
13 printf(" White:%d(decimal) %x(hex)\n" , WHITE, WHITE);
```

Java-Arrays vs. C-Arrays

- ▶ Arrays in C Sind ähnlich zu Arrays in Java
- ▶ Länge in **Bytes** kann mit **sizeof** abgefragt werden
- ▶ Für echte Länge: Division mit der Größe eines Elements des Arrays → ebenfalls mit **sizeof**

```
1 Color rainbow [] = {RED, YELLOW, GREEN, CYAN, BLUE,  
    MAGENTA};  
2 int length = sizeof(rainbow)/sizeof(int);  
3  
4 //Verwendung:  
5 printf("Der Regenbogen hat %d Farben\n", length);
```

C-Arrays im Detail

- ▶ Werte eines Arrays werden hintereinander im Speicher abgelegt
- ▶ Natürlich ist es in C genauso wie in Java möglich mehrdimensionale Arrays anzulegen
- ▶ Array selbst ist nur die Adresse des ersten Elements
- ▶ Zur Laufzeit weiß das C-Programm nicht mehr, dass es ein Array ist
 - ▷ Bei **sizeof** legt Compiler eine Tabelle an
 - ▷ Diese wird nach dem Übersetzen gelöscht
 - ▷ Dort ist unter dem **Namen** die Länge des Arrays vermerkt

```
1 Color rainbow [] = {RED, YELLOW, GREEN, CYAN, BLUE,  
    MAGENTA};  
2 int length = sizeof(rainbow)/sizeof(int);
```

struct

```
1 struct name{  
2     datentyp1 var1;  
3     datentyp2 var2;  
4     ...  
5 };
```

- ▶ Benutzerdefinierter komplexer Datentyp
- ▶ (errinert an Java Klassen)
- ▶ Dabei verwendet man das Schlüsselwort **struct**
- ▶ Auch **typedef** möglich, ähnlich wie enum
- ▶ **struct** Konstrukt so groß, wie die Summe der Größen seinen Membervariablen
- ▶ Semikolon am Ende!

Übungsaufgabe: Verwendung von struct

Eine Handvoll Farben mit *enum* ist zu begrenzt. Wir wollen die einzelnen RGB Farbkanäle *red*, *green* und *blue* gezielt verändern und damit alle Farben definieren, die es gibt.

- ▶ Deklariert ein *struct* **colorDefinition** mit den „Membervariablen“ **red**, **green** und **blue** (unsigned char : 8 Bit, 0..255)
- ▶ definiert die Farben **black**, **blue** und **white** als „Objekte“.
- ▶ Gibt den R-Wert von **black** am Bildschirm aus

4!

Lösung: struct

```
1 typedef struct colordef{
2     unsigned char red;
3     unsigned char green;
4     unsigned char blue;
5 } colorDefinition;
6
7
8 colorDefinition black={0,0,0};
9 colorDefinition white={255, 255, 255} ;
10 colordefinition blue={.blue=255};
11
12 //Verwendung:
13 printf("%d", black.red);
```

```
1 union name{  
2     datentyp1 var1;  
3     datentyp2 var2;  
4     ...  
5 };
```

- ▶ Wird genauso wie ein Struct deklariert
- ▶ Dabei verwendet man jedoch das Schlüsselwort **union**
- ▶ Alle Variablen sind Bezeichner für dieselbe Speicherstelle
- ▶ somit ist der **union** Konstrukt nur so groß, wie seine größte Variable (Platzsparend!)
- ▶ Damit können wir **colorDefinition** erweitern, sodass wir wahlweise RGB-Angaben oder HEX-Werte speichern können.

Union: Erweiterung des Color-Types

```
1 typedef union color{
2     struct rgb{
3         unsigned char red;
4         unsigned char green;
5         unsigned char blue;
6     }rgb;
7     int hex;
8 } Color;
9
10 Color schweinchenRosa={0xffb6c1};
11 Color navy={0,0,128}
12
13 //Verwendung
14 printf("B-Wert der Navy: %d\n", navy.rgb.blue);
```

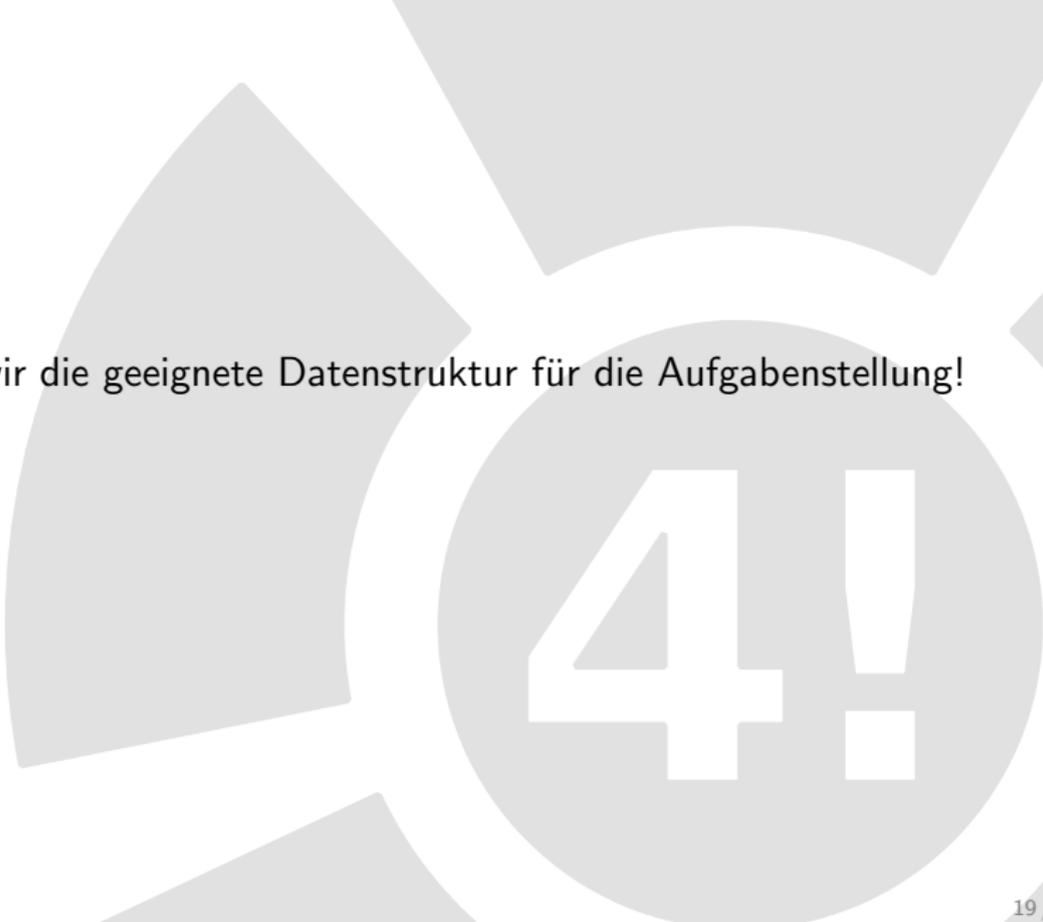
Zusammenfassung: enum, Arrays

- ▶ **enum color** : Neu definierter Datentyp mit einer Auflistung von möglichen Werten.
- ▶ **typedef enum color {...} Color** : das Wort *Color* kann als Datentyp anstelle von *enum color* verwendet werden
- ▶ **Color rainbow[6]** : ein Array, das 6 Elemente vom Typ Color enthalten kann
- ▶ **Color frankreich[]={BLUE, WHITE, RED}** : ein Array, das 3 Elemente vom Typ Color enthält
- ▶ **sizeof(frankreich)/sizeof(int)** : die Länge unseres Arrays (hier:3)

Zusammenfassung: struct, union

- ▶ **struct stru**{**int a; char b;**}; Neu definierter Komplexer Datentyp *struct stru* mit int und char nacheinander
- ▶ **struct stru var1**={**1,'x'**}; : Verwendung von struct, beide Variablen werden gesetzt
- ▶ **union uni**{**int a; char b;**}; Neu definierter Datentyp *union uni*, bei der die Elemente wahlweise int oder char enthalten können
- ▶ **union uni var2**={**.a=1**}; : Verwendung von union, nur eine Variable darf gesetzt werden

Verwenden wir die geeignete Datenstruktur für die Aufgabenstellung!



4!

Übungsaufgabe: Arrays wie in Java



```
1 Color rainbow [] = {RED, YELLOW, GREEN, CYAN, BLUE,  
   MAGENTA};  
2 int length = sizeof(rainbow)/sizeof(int);
```

Angenommen, wir wollen eine Funktion **printColor** benutzen, welche die einzelnen Farben im Array auf dem Terminal ausgibt. Die Funktion **printColor** hat dabei folgende Signatur

```
1 void printColor(Color farben [], int length);
```

```
1 //Regenbogen
2 color rainbow [] = {RED, YELLOW, GREEN, CYAN, BLUE,
   MAGENTA};
3 int length1 = sizeof(rainbow)/sizeof(int);
4
5 //Frankreichfahne
6 color frankreich [] = {BLUE, WHITE, RED};
7 int length2 = sizeof(fr frankreich)/sizeof(int);
```

- ▶ Bei mehreren Arrays, die an **printColor** übergeben werden, muss man die Arrays jedesmal anlegen, die Länge ermitteln, in einer separaten Variable speichern und merken, etc... unübersichtlich!
- ▶ Wir vermissen java Arrays und `.length`!

Implementierung: Arrays wie in Java

```
1 typedef struct colorarray{
2     const int length;
3     color content[9];
4 } colorArray;
5
6 //Ein colorListe struct anlegen,
7 colorArray rainbow={
8     .length= 9,
9     .content= {RED, YELLOW, GREEN, CYAN, BLUE, MAGENTA, RED,
10                BLUE, YELLOW}
11 };
12
13 //Laenge des Arrays
14 int array_length = rainbow.length
```