C-Kurs 2012

Datentypen

Theresa Enghardt
theresa@freitagsrunde.org
basierend auf den Folien von Eugen Rein

http://wiki.freitagsrunde.org

10. September 2012

Inhaltsverzeichnis

Aufzählungen (Enums)

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

- Neu definierter Datentyp: Wochentag als enum mit 7 möglichen Werten
- Der Compiler wandelt die Elemente des Enum in Integer um

```
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  }
```

Aufzählungen (Enums)

- werden auch als Aufzählungstypen bezeichnet
- Intern als Ganzzahlen gehandelt
- Elemente nacheinander durchnummeriert
- Beispielsweise lassen sich die Wochentage gut repräsentieren

Boolean repräsentiert durch Enums

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

Eine Idee?

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

Boolean als Enum

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

```
1 enum bool{
2 FALSE = 0,
3 TRUE = 7
4 };
```

Enum-Boolean mit typedef

- Verwendet kann es werden mit

```
1 enum bool flag = TRUE;
```

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

```
1 typedef enum bool{
2 FALSE,
3 TRUE
4 } boolean;
5
6 boolean flag = TRUE;
```

Colors mit Enums

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

Lösung der Übungsaufgabe

```
1  typedef enum color_t {
2    WHITE = 0xFFFFFF,
3  BLACK = 0x000000,
4  RED = 0xFF0000,
5  YELLOW = 0xFFFF00,
6  GREEN = 0x00FF00,
7  CYAN = 0x00FFFF,
8  BLUE = 0x0000FF,
9  MAGENTA = 0xFF00FF
10 } color;
```

Java-Arrays vs. C-Arrays

- 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
  $\rightarrow$ ebenfalls mit **sizeof**

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

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

Frage: Übergabe Arrays an Funktionen

Welches Problem tritt bei der Übergabe des Arrays an eine Funktion auf?

Übungsaufgabe: Übergabe Arrays an Funktionen

Nun wollen wir eine Funktion **printColors** schreiben, welche die einzelnen farben im Array auf dem Terminal ausgibt. Die Funktion **printColors** hat dabei folgende Signatur

```
1 void printColors(color colors[], int colorsLength);
```

- Funktion **printColors** wird bei mehreren Arrays, die übergeben werden schnell unübersichtlich $\rightarrow$ Sprachkonzept der **structs**
- Ähnlich zu Enums

Implementierung: Arrays mit länge

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

Übungsaufgabe: Colors Revisited

Farben als Enums zu definieren, war zwar ein nettes Einsteigerbeispiel, bewährt sich in der Praxis aber nicht sonderlich, weil man die einzelnen Farbkanäle nicht gezielt verändern kann. Deklariert ein struct color mit den „Membervariablen“ **red**, **green** uns **blue** und definiert unsere altbekannten Variablen **black**, **white**, **red**, **yellow**, **green**, **cyan**, **blue**, und **magenta** als „Objekte“.

- Wird genauso wie ein Struct deklariert
- Dabei verwendet man jedoch das Schlüsselwort **union**
- Alle Variablen sind Bezeichner für dieselbe Speicherstelle
- Damit können wir **color** erweitern, sodass wir RGB-Farben als auch HEX-Werte speichern können.

Erweiterung des Color-Types

```
1 typedef union color_t {
2   struct rgb_t {
3     unsigned char red;
4     unsigned char green;
5     unsigned char blue;
6   } rgb;
7   int hex;
8 } color;
```