C-Kurs 2011

Datentypen

Eugen Rein

Technische Universität Berlin

14. September 2011

Inhaltsverzeichnis

Wiederholung Enums

- werden auch als Aufzählungstypen bezeichnet
- Intern als Ganzzahlen gehandelt
- Elemente nacheinander durchnummeriert
- Beispielsweise lassen sich die Wochentage gut repräsentieren
  $\rightarrow$ Vortrag 1 von Paul

Boolean repräsentiert durch Enums

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

Eine Idee?

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!

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

C-Arrays im Detail

- Werte eines Arrays werden hintereinander im Speicher abgelegt
- 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

Gegeben ist folgendes Code-Fragment, das in eine Funktion **printRainbow** ausgelagert werden soll.

```
1 int i = 0;
2 for(i; i<sizeof(rainbow)/sizeof(int); i++){
3     attron(COLOR_PAIR(rainbow[i]));
4     addstr("\u2588");
5     attroff(COLOR_PAIR(rainbow[i]));
6   }
```

Die Funktion **printRainbow** hat dabei folgende Signatur

```
1 void printRainbow(int rainbow[], int rainbowLength);
```

- Funktion **printRainbow** wird bei mehreren Arrays, die übergeben werden schnell unübersichtlich
- C bietet rudimentär auch Klassen an
  $\rightarrow$ Sprachkonzept der **structs**
- Ähnlich zu Enums
- Damit können wir Arrays wie in Java nachbauen

Implementierung: Arrays wie in Java

```
1  typedef struct rainbow{
2    const int length;
3    int color[6];
4  } Rainbow;
5  ...
6  /*Konstruktor aufrufen*/
7  Rainbow rainbow={
8    .length= 6,
9    .color= {COLOR_RED, COLOR_YELLOW, COLOR_GREEN,
         COLOR_CYAN, COLOR_BLUE, COLOR_MAGENTA}
10 };
11
12 /*Laenge des Arrays*/
13 int array_length = rainbow.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{
2   struct rgb{
3     unsigned char red;
4     unsigned char green;
5     unsigned char blue;
6   } rgb;
7   int hex;
8 } Color;
```