6
Arrays in C

Jörn Loviscach

Versionsstand: 25. September 2014, 18:39

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen beim Ansehen der Videos:
http://www.j3L7h.de/videos.html

Bitte hier notieren, was beim Bearbeiten unklar geblieben ist

1 Grundlagen

Nach `bool` und den diversen Zahlentypen kommt nun die erste komplexe Datentyp („Datenstruktur“): das Array, im Deutschen auch Feld genannt. Es speichert n oder $n \times m$ oder $n \times m \times p$ oder ... Elemente eines festen Datentyps, vergleichbar mit einer Liste, einem Setzkasten oder einem Wohnsilo:

[1]

Jede Zelle eines Arrays lässt sich mit Hilfe ihres Index bzw. ihrer Indizes finden. In C und seinen Nachfolgesprachen zählt man allerdings ab null, nicht ab eins:

[2]

Die Dimension des Arrays sagt, wie viele Indizes man benötigt, um eine Zelle anzugeben.

Wozu das?

[3]

So baut man ein Array aus 4 mal 5 Zellen für Integer-Zahlen:

[4]

So wird in eine Zelle geschrieben:

[5]

So wird aus einer Zelle gelesen:

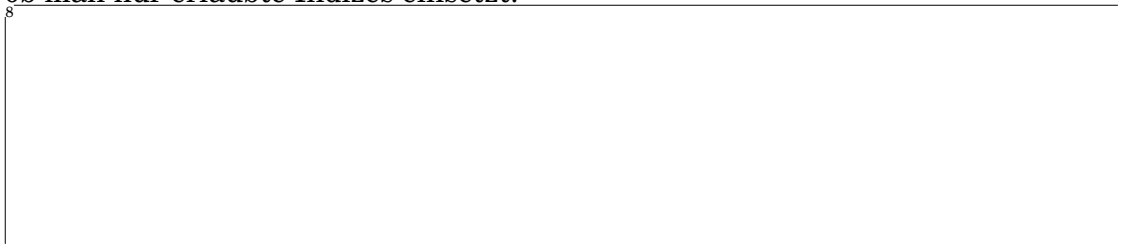
[6]

So wird der Inhalt einer Zelle um eins erhöht:

[7]


Demo im Debugger. Genau hingucken mit den eckigen Klamern!

Achtung: C und C++ prüfen der Einfachheit und der Geschwindigkeit halber *nicht*, ob man nur erlaubte Indizes einsetzt:

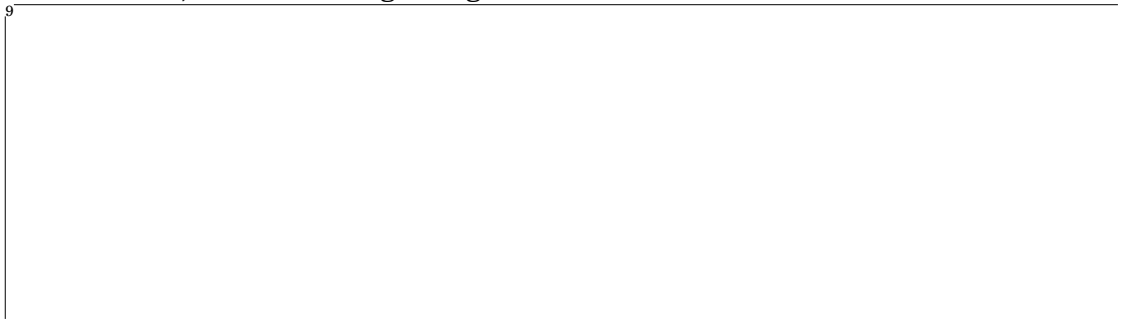
[8]


Wenn man also beim Programmieren nicht aufpasst, kann es passieren, dass man außerhalb der Grenzen des Arrays im Speicher liest oder schreibt. Damit kann man sich ganz andere Teile seiner Daten oder seines Programms zerschießen. Das sorgt für schwer zu findende Fehler. Außerdem ist dies der wesentliche Grund für Sicherheitslücken. Zum Beispiel kann ein Angreifer so viele Daten eingeben, dass sie in den Speicher hinter dem Array geschrieben werden (Buffer Overflow).

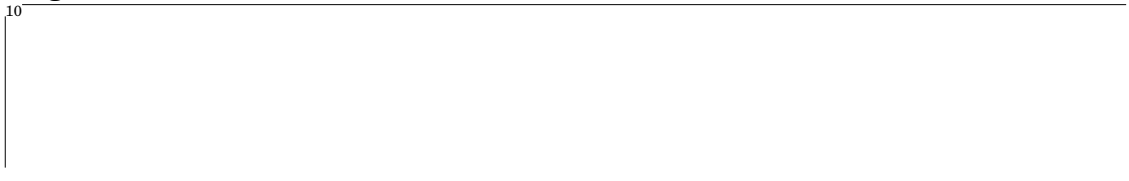
In C++ und vor C99 auch in C muss die Größe eines Arrays mit einer Konstanten angegeben sein. Sie kann nicht erst im Programm berechnet werden, zum Beispiel aus dem Umfang der Eingangsdaten. Es gibt später Wege, um dieses Problem zu umgehen.

2 Initialisierung und Zuweisung

Nicht-statische Variablen (ob Arrays oder nicht) enthalten in C und C++ zu Beginn die Zahlen, die gerade zufällig an der Stelle im Speicher stehen. Das spart dem Rechner Zeit, ist aber nervig und gefährlich. Man kann die Werte zu Fuß setzen:

[9]


Es geht aber noch einfacher:

[10]


Wenn man alle Elemente angibt, kann der Compiler sich die Größe selbst ausrechnen:

```
11
```

Wenn man nicht alle alle Elemente angibt, setzt der Compiler die ungenannten auf null:

```
12
```

Auf diese Weise kann man ein Array einfach mit Nullen füllen:

```
13
```

In zwei Dimensionen usw. sieht das entsprechend aus:

```
14
```

Achtung: Diese Schreibweisen mit Schweifklammen klappen nur beim Initialisieren eines Arrays, danach nicht mehr!

Leider ist es nicht ganz so leicht, in C und C++ einem Array komplett ein anderes Array zuzuweisen. Dies meldet der Compiler als Fehler:

```
15
```

Man kann langweilig mit einer `for`-Schleife kopieren:

```
16
```

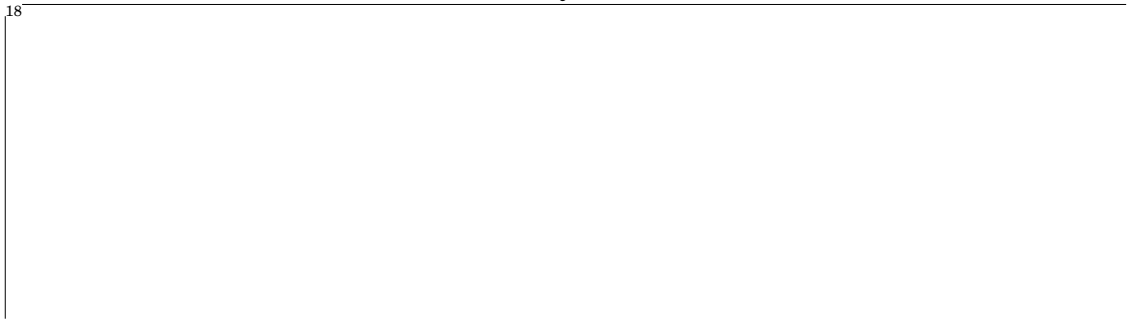
Oder die in `<string.h>` deklarierte Funktion `memcpy` nutzen:

[17]


Der hintere Parameter sagt, wie viele Bytes im Speicher kopiert werden sollen. Mit Hilfe des Operators `sizeof` kann man sein Programm dabei plattformneutral und besser lesbar halten. (Dieser Operator ist etwas eigenwillig: Etwas Konkretes wie `sizeof 2.34` geht ohne Klammern; Typnamen müssen dagegen wie `sizeof (double)` in Klammern stehen.)

3 Übergabe und Rückgabe von Arrays

In eine Funktion hinein kommt ein Array in C und C++ recht einfach:

[18]


Die Größe der allerersten Dimension ist nicht nötig, um auf das Array zugreifen zu können. Man darf in dem formalen Parameter in der Funktionsdeklaration für die vorderste Dimension leere Klammern `[]` schreiben.

Bei dieser Übergabe an die Funktion wird das Array allerdings nicht wie eine normale Variable kopiert. Vielmehr ist alles, was die obige Funktion dem `x` antut, in der aufrufenden Funktion beim `a` sichtbar. Beides ist ein und dieselbe Sammlung an Daten, nur unter verschiedenen Namen angesprochen.

Ein Array aus einer Funktion zurückzugeben, ist dagegen in C und C++ so nicht vorgesehen. Das würde schon daran scheitern, dass Arrays nicht zugewiesen werden können und nicht automatisch kopiert werden. (Das zeige ich lieber gar nicht erst mit einem – zwangsläufig falschen – Codebeispiel.) Als Ausweg übergibt man der Funktion überlicherweise ein Array, das sie dann mit Werten füllt. Das klappt, weil das Array ja nicht in ein neues Array kopiert wird, sondern in der Funktion einfach nur unter anderem Namen lebt:

[19]