# 10
# Die Programmiersprache C99: Zusammenfassung

Jörn Loviscach

Versionsstand: 25. September 2013, 18:07

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 Compilation, Präprozessor

C-Compiler und C++-Compiler übersetzen jede .c-Datei bzw. .cpp-Datei einzeln. Die Ergebnisse (Objektdateien) werden dann vom Linker zu einer ausführbaren Datei zusammengebunden. Der Compiler braucht vor der Verwendung jeder Funktion eine Deklaration (oder sogar Definition). Die Deklarationen der in anderen Dateien genutzten Funktionen sowie die Definitionen von in anderen Dateien genutzten Konstanten, Inline-Funktionen und eigenen Typen `struct` und `enum` lagert man typischerweise in .h-Dateien (Header) aus – zumindest für die Funktionen/Definitionen/Typen, die anderswo im Programm genutzt werden sollen.

Der Übersetzungsprozess sieht so dann aus:

```
1
```

Kompiliert werden die .c- oder .cpp-Dateien, aber der Präprozessor holt mit `#include` die Header-Dateien dazu:

```
2
```

Mit „Include Guards“ in den Header-Dateien sorgt man dafür, dass der Compiler sie für jede .c- und .cpp-Datei nur einmal sieht:

```
3
```

Bei fortgeschritteneren Programmen kann es nämlich passieren, dass man in derselben .c- oder .cpp-Datei zwei Header-Dateien einbindet, die beide ihrereseits dieselbe Header-Datei einbinden:

```
4
```

Dann der Compiler diese letztere Header-Datei insgesamt zweimal für die eine .c- oder .cpp-Datei sehen. Ohne Include Guards ist das ein Problem, wenn die

Header-Datei zum Beispiel dies enthält:

5

Mit `static` deklarierte Funktionen und globale Variablen sind nur in der aktuell compilierten .c- oder .cpp-Datei sichtbar. (Eine `static`-Variable *innerhalb* einer Funktion, also lokal, lebt auch für die gesamte Dauer des Programms, ist aber nur in der Funktion sichtbar.) Will man eine globale Variable in anderen .c- oder .cpp-Dateien nutzen, muss man sie dort – zum Beispiel per Header-Datei – mit `extern` deklarieren.

6

# 2 Typen, Operatoren und Präzedenz

Dies waren die grundlegenden Typen:

7

Konstante Zahlen gibt man so an:

8

Bei Rechnungen mit ganzen Zahlen arbeiten C und C++ meist mindestens mit `int`. Aber oft reicht das nicht. Dann hilft ein Cast:

9

Ebenso muss man beim Dividieren vorsichtig sein:

10

Dies sind die meisten der Operatoren von C und C++, aufgelistet nach Präzedenz (Rangfolge nach Art von „Punktrechnung vor Strichrechnung“:

11

Runde Klammern können benutzt werden, um eine andere Reihenfolge zu erreichen:

12

Aufzählungen vergeben Namen für ausgewählte Werte ganzzahliger Variablen:

13

Aus allen Typen kann man ein- oder mehrdimensionale Arrays bilden:

14

Wie nichtstatische lokale Variablen sind auch nichtstatische lokale Arrays zu Beginn mit beliebigen Werten gefüllt. Fast immer ist eine Initialisierung nötig:

15

Arrays aus `char` werden in C als Zeichenketten [character strings] aufgefasst. Eine Null zeigt das Ende an. Feste Zeichenketten gibt man mit doppelten Anführungszeichen an. Nicht mit einzelnen Zeichen verwechseln!

16

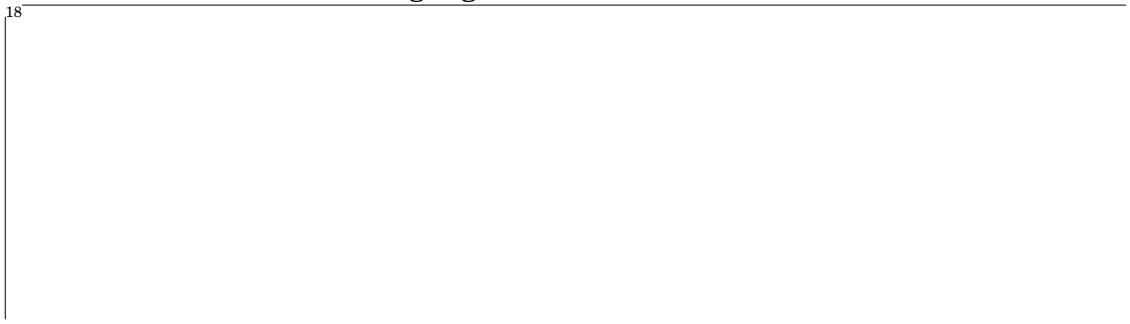
Aus allen Typen kann man Verbünde = Strukturen bilden:

17

# 3 Verzweigungen und Schleifen

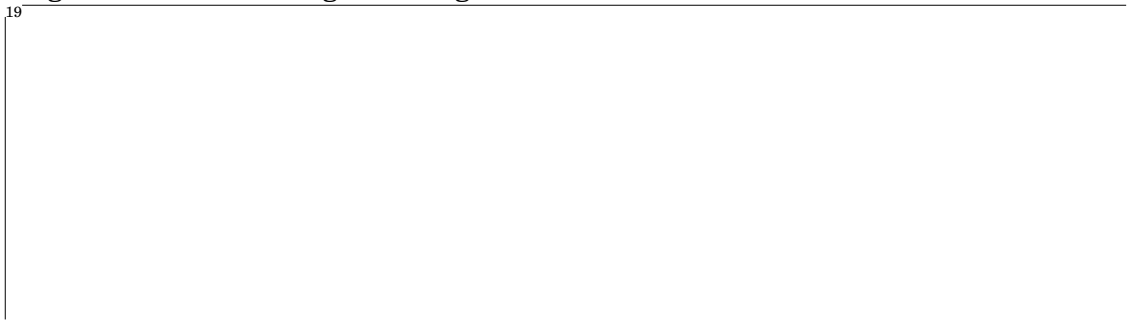
Die übliche Verzweigung ist die mit `if` und ggf. `else`. In dem `else`-Zweig bringt man oft noch weitere Verzweigungen unter:

18



Mit der anderen Verzweigung `switch` kann man in einem Rutsch eine Liste an Möglichkeiten für eine ganzzahlige Variable abarbeiten:

19



C und C++ kennen drei Schleifen: `while` mit dem Vergleich am Anfang, `do...while` mit dem Vergleich am Ende und `for` mit einer Zählvariablen:

20

Alle drei Schleifen können mit `break;` abgebrochen werden. Und in allen drei Schleifen kann man mit `continue;` den aktuellen Schleifendurchlauf abbrechen und zum nächsten springen.

# 4 Funktionen, Standardbibliothek

Funktionen kommen in C und C++ auf drei Arten vor: Deklaration, Definition und Aufruf.

21

Vor jedem Aufruf einer Funktion muss der Compiler in der jeweiligen .c- oder .cpp-Datei eine Deklaration der Funktion gesehen haben; die Funktion kann auch vorher definiert sein. In der Deklaration steht `void` statt eines Typs, wenn die Funktion nichts zurückliefert bzw. keine Werte (Parameter bzw. Argumente) entgegennimmt.

Mit `return ...;` verlässt man eine Funktion und bestimmt gleichzeitig, welches Resultat (Rückgabewert) die Funktion haben soll.

Eine Funktion, deren Rückgabetyp `void` ist, kann bis zu ihrem Ende durchlaufen oder aber vorher mit einem leeren `return;` beendet (verlassen) werden. Funktionen, die laut Deklaration einen Wert zurückliefern, sollten nur mit `return ...;` verlassen werden, sonst kommt Unsinn als Ergebnis.

Wie überall in C und C++ bleiben auch bei Funktionen die in den Schweifklammern eingeführten Variablen nach außen unsichtbar (lokale Variablen). Ebenso bleiben die Argumente nach außen unsichtbar.

Die meisten mathematischen Funktionen sind in `<math.h>` deklariert. Darunter finden sich zum Beispiel:

22

Achtung 1: Der Absolutbetrag für `int` heißt `abs` und ist in `<stdlib.h>` deklariert. Achtung 2: `^` ist *nicht* die Potenzfunktion, auch nicht `**`. Achtung 3: Die Potenzfunktion `pow` ist zu schwerfällig für Ganzzahlen und für Quadrate von Gleitkommazahlen.

In `<stdio.h>` sind Ein- und Ausgabefunktionen deklariert:

23

In `<string.h>` sind z. B. Funktionen zum Kopieren, Vergleichen und Aneinanderhängen von Zeichenketten deklariert:

[24]

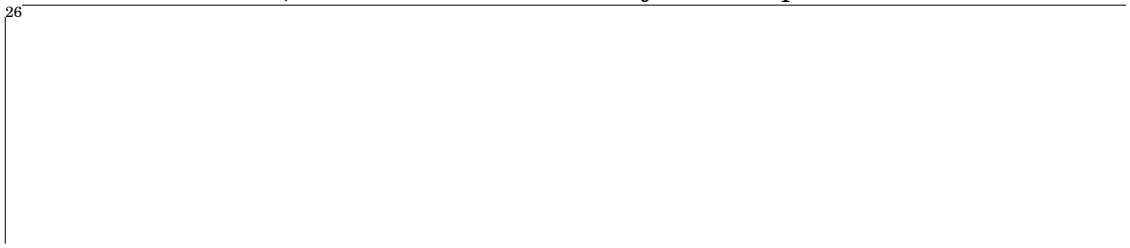
Jedes C- oder C++-Programm benötigt genau eine Funktion `main`. Das ist die Funktion, die beim Start des Programms aufgerufen wird. Diese Funktion muss eine dieser beiden Signaturen haben:

[25]

# 5 Zeiger und Dynamischer Speicher

C und C++ erlauben, mit der Adresse eines Objekts im Speicher zu arbeiten:

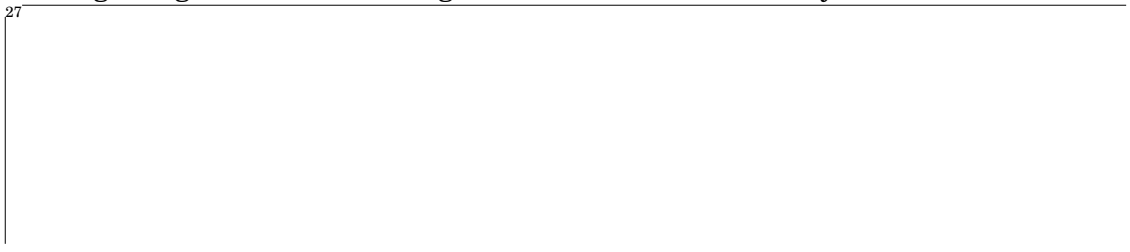
[26]



Zeigervariablen, die ungültig sind, sollten auf `NULL` gesetzt werden (deklariert in `<stddef.h>`).

Die Namen von Arrays verhalten sich weitgehend wie Zeiger auf den vordersten Eintrag. Umgekehrt können Zeiger wie die Namen von Arrays verwendet werden:

[27]



Zeigerarithmetik: Zu/von Zeigern dürfen ganze Zahlen addiert/subtrahiert werden. Das ändert die Zeigern nicht in Schriten von Bytes, sondern in Schritten der Größe

des jeweiligen Datentyps:

[28]

Mit der in `<stdlib.h>` deklarierten Funktion `malloc` fordert man eine gewünschte Zahl Bytes an Speicherplatz an. Wenn man den nicht mehr benötigt, sollte man ihn mit `free` wieder freigeben:

[29]