Statements (auf Deutsch: Befehle)

Allgemeines:

- Zusammengehörende Statements (einer Funktion, einer Schleife, eines `if`, ...) stehen in `{ }`.
- Am Anfang jedes `{ }` können Deklarationen stehen.
- Jedes Statement endet entweder mit einem `;` oder mit einem `{ }`-Block. Deklarationen usw. enden ebenfalls immer mit einem `;`.
- Daher gilt:
  - ***Nach*** einer `}` steht ***nie*** ein `;`[1].
  - ***Vor*** einer `}` steht ***immer*** ein `;`, ***außer*** bei leeren `{}` und bei mehreren `}` hintereinander.
- Der Rumpf eines `if`, eines `else` oder einer Schleife besteht entweder aus einem ***einzigen*** Befehl ***ohne*** `{ }` (beispielsweise `if (i < 0) i = -i;`), oder aus ***beliebig vielen*** Befehlen in `{ }`.
- Ich empfehle ***auch bei nur einem Befehl*** immer `{ }`, wenn der Befehl in der Zeile drunter und nicht gleich dahinter steht: Fügt man später einen zweiten Befehl dazu, vergißt man leicht auf die `{ }`, und dann tut das Programm nicht das, was die Einrückung vortäuscht!
- Weiters empfehle ich bei leerem Schleifen-Rumpf immer `{ }` statt einem einzelnen `;`: Bei `for (...) {}` sieht man viel deutlicher, was gemeint ist, als bei `for (...) ;` .
- Ein ganz böser Fehler ist ein `;` zwischen `)` und `{` bei `if`, `while` und `for`: Das Statement endet beim `;`, und die Befehle in den `{ }` werden unabhängig von der Bedingung immer genau einmal ausgeführt!

expr`;`

expr (Expression, auf Deutsch: Ausdruck) ist ein beliebiger Ausdruck (eine "Rechnung"), üblicherweise eine Zuweisung wie `j = 2 * i;` oder ein Funktionsaufruf wie `printf(...);`.

`if (`*expr*`)` *statements*

`if (`*expr*`)` *statements* `else` *statements*

`if (`*expr*`)` *statements* `else if (`*expr*`)` *statements* `...  else` *statements*

expr ist eine Bedingung, d. h. ein Ausdruck, der *wahr* oder *falsch* liefert, üblicherweise ein Vergleich. Bei wahr werden die *statements* hinter dem `if` ausgeführt, bei falsch jene hinter dem `else`, bei `else if`-Konstrukten jene hinter dem ***ersten*** wahren *expr*. Es wird immer nur ***ein*** Zweig ausgeführt, dann geht es mit dem nächsten Befehl nach dem ganzen `if-else`-Konstrukt weiter.

Achtung: Ein `else` gehört (auch wenn die Einrückung anders ausschaut) immer zum ***letzten*** `if` ohne `else` auf gleicher `{ }`-Ebene. Im Zweifelsfall `{ }` verwenden!

Achtung: Wie schon besprochen, muß eine Bedingung in C nicht unbedingt ein Vergleich sein: Als *expr* ist jede Rechnung erlaubt, die einen einzelnen Wert (keine Struktur o. ä.) als Ergebnis liefert: `0` oder `'\0'` oder `NULL` gilt als falsch, jeder von 0 verschiedene Wert als wahr!

---

[1] Außer bei ganz seltenen `struct`-Typdefinitionen.

`while (`*expr*`)` *statements*

expr ist wieder eine Bedingung. Liefert sie wahr, werden die *statements* in der Schleife ausgeführt. Danach wird *expr* wieder geprüft usw.. Liefert *expr* falsch, wird mit dem nächsten Befehl nach der Schleife weitergemacht.

expr wird ***vor*** der Schleife geprüft: Liefert *expr* von Anfang an falsch, werden die *statements* ***kein einziges Mal*** ausgeführt.

`while (1)` ergibt eine Endlosschleife.

Notiere dir die Beispiele!

`do` *statements* `while (`*expr*`);`

Wie `while`, aber die Bedingung *expr* wird ***nach*** dem Schleifendurchlauf geprüft. Die *statements* werden daher ***mindestens einmal*** ausgeführt!

Achtung: Im Vergleich zum `until` in PASCAL und anderen Programmiersprachen ist die Bedingung hier logisch invertiert: Die Schleife läuft, ***solange*** *expr* wahr ist, und nicht ***bis*** *expr* wahr ist!

`for (`*expr1*`;` *expr2*`;` *expr3*`)` *statements*

`for` wird für ***Zählschleifen*** verwendet: *expr1* ist die Initialisierung (Zuweisung des Anfangswertes), *expr2* ist die Bedingung (Test auf den Endwert), *expr3* ist das Weiterzählen (üblicherweise ein `++`).

Ein `for` ist definiert als

```
expr1;
while (expr2) {
  statements
  expr3;
}
```

expr1 wird daher einmal ***vor*** der eigentlichen Schleife ausgeführt, die Bedingung *expr2* wird vor jedem Durchlauf geprüft (daher werden die *statements* kein einziges Mal ausgeführt, wenn *expr2* gleich am Anfang falsch ergibt), und *expr3* wird bei jedem Durchlauf ***nach*** den *statements* aber ***vor*** dem erneuten Test der Bedingung *expr2* ausgeführt.

Notiere dir die Beispiele!

Jeder Teil kann weggelassen werden, eine weggelassene Bedingung gilt als wahr: `for (;;)` ist eine Endlosschleife (außer, sie wird mit `break;` verlassen).

`return;` **oder** `return` *expr*`;`

Kehrt aus der gerade laufenden Funktion in die aufrufende Funktion zurück, und zwar bei `return;` ohne einen Returnwert (bei `void`-Funktionen) und bei `return` *expr*`;` mit dem Returnwert *expr* (bei Funktionen mit einem "echten" Return-Typ wie z. B. `int`)[2].

`return` braucht keine `( )` um *expr*!

`break;`

Verläßt die innerste gerade laufende Schleife: Das Programm macht hinter der `}` der unmittelbar umgebenden Schleife (oder des `switch`-Befehls) weiter.

---

[2] Für das Beenden von `main` ist die Programmende-Funktion `exit` empfohlen und kein `return`!

`continue;`

Beendet den gerade laufenden Schleifendurchlauf der innersten gerade laufenden Schleife: Das Programm macht mit dem nächsten Durchlauf der unmittelbar umgebenden Schleife weiter (bei `while`-Schleifen mit dem Test der Bedingung, bei `for`-Schleifen mit dem Weiterzählen).

`switch (`*expr*`)` *caselist* **und** `case` *const*`:` **und** `default:`

expr wird ausgerechnet (muß einen `char`, einen `int`, oder einen Enumerationstyp liefern), und die Ausführung springt zu dem Befehl unmittelbar hinter jenem `case`, dessen *const* gleich dem Wert von *expr* ist. Dort wird bis zu einem `break;` oder dem Ende der *caselist* weitergearbeitet, das `break;` springt ebenfalls hinter das Ende der *caselist*.

Notiere dir das Beispiel!

Paßt kein `case` zum Wert von *expr*, werden die Befehle hinter `default:` ausgeführt. Gibt es kein `default:`, wird in diesem Fall beim nächsten Befehl hinter dem gesamten `switch`-Konstrukt weitergearbeitet. Im Unterschied zu anderen Programmiersprachen wird ***kein*** Fehler gemeldet, wenn kein Fall zutrifft (man sollte aber aus Stil- und Sicherheitsgründen trotzdem immer ein `default:` schreiben; wenn es nie passieren darf, dann eben mit einer Fehlermeldung oder einem `abort()`!)!

Achtung: Im Unterschied zu ähnlichen Befehlen in anderen Programmiersprachen ***fällt das*** `switch` ***in den nächsten*** `case` ***durch*** (schlimmstenfalls bis zur `}` am Ende des gesamten `switch`)**,** ***wenn man das*** `break` ***vergißt!***

Achtung: Die *const* beim `case` müssen Konstante oder Rechnungen mit fixem Ergebnis sein, keine Variablen oder Rechnungen mit Variablen!

Man kann im `case` nicht mehrere Werte (z. B. mit `,` getrennt) angeben, und auch keine von-bis-Bereiche[3], aber man kann mehrere `case` hintereinander schreiben: Die Befehle dahinter gelten dann für alle `case` davor.

label: *statement*; **und** `goto` *label*;

label (auf Deutsch: Marke) ist ein beliebiger Name. Bei einem `goto` springt die Programmausführung zu jenem *statement*, vor dem das angegebene *label* steht (innerhalb einer Funktion, nicht zwischen Funktionen!).

Dieser Mechanismus ergibt besonders fehleranfällige und schwer verständliche Programme und ist daher in der Schule (und den meisten Firmen) ***verboten***!

[3] `gcc` kann von-bis-Bereiche mit `....`