

# **C Coding Style**

**Prof. Dr.-Ing. Franz Korf**

[Franz.Korf@haw-hamburg.de](mailto:Franz.Korf@haw-hamburg.de)

## Einleitung

### Für wen ist dieser C Coding Style gedacht?

- Er richtet sich an Studierende, die sich in C einarbeiten und schon Erfahrungen in anderen Programmiersprachen haben.
- Viele Aspekte wie zum Beispiel die Entwicklung großer C Programmpakete oder firmenspezifische Coding Styles werden nicht abgedeckt.
- Der folgende, sehr kurze Coding Style basiert auf den Ausarbeitungen von Wolfgang Fohl und ist für die Situation im Praktikum gedacht. Es sind wenige Regeln, aber diese müssen Sie einhalten!

### Quelle:

- John Reekie: C Coding Style, University of Technology Sydney, 1993
- C Coding Style von Wolfgang Fohl
- Eine Sammlung von Coding Styles findet man hier:  
<http://www.chris-lott.org/resources/cstyle/>

## Drei goldene Regeln

### Keep it simple.

- Diese Regel besagt zum Beispiel: Verwenden Sie keine komplexen Sprachkonstrukte, wenn einfache auch zum Ziel führen.

### Don't be clever.

- Diese Regel besagt zum Beispiel: Verwenden Sie nur bei Bedarf die Tricks, die Sie vielleicht gerade erst gelernt haben. Nicht jeder kennt diese Tricks.

### Be explicit – write your program for people.

- C Programme werden nicht nur von C Experten gelesen und gewartet. Schreiben Sie Programme so, dass andere Personen dieses leicht lesen und verstehen können – dann können Sie Ihr Programm auch nach einem Jahr noch leicht lesen und verstehen.
- Verständliche Programme beeindrucken, unverständliche Programme schrecken ab.

**Der folgende Coding Style dient als Richtlinie. Verbesserungen und Modifikationen, die den goldenen Regeln folgen, sind stets möglich.**

# Bezeichner

## Anforderungen an einen Bezeichner:

- Ein Bezeichner spiegelt den Zweck der Variablen, des Typs, des Makros ... wieder.
- Ein Bezeichner sollte möglichst präzise sein.

## Eine Konvention für Bezeichner:

- **Variablennamen und Funktionsnamen** müssen mit einem Kleinbuchstaben beginnen.
- **Konstantennamen** dürfen keine Kleinbuchstaben enthalten.
- Der Name von struct-**Typen** sowie von Typen, die Sie mit typedef definiert haben, muss mit einem Großbuchstaben beginnen.
- Alle Namen müssen nach dem selben Prinzip aufgebaut werden (s. zum Beispiel POSIX). Die Wahl eines Bezeichner hängt von seinem Scope ab.
- Bezeichner von Makros bestehen nur aus Großbuchstaben und “\_”. So wird ihre besondere Behandlung durch den Präprozessor hervorgehoben.

## Bezeichner (Fortsetzung)

| Beispiel                                   | Typische Verwendung   |
|--|---|
| i, j, k, ch                                | Lokale Variablen mit einem kleinen Scope (z.B. kleiner Block, oder kleine Funktion). i, j, k sind zum Beispiel Zähler, ch ist oftmals ein Puffer für ein Zeichen. |
| remove, add, size                          | Lokale Variablen, lokale Funktionen, oft verwendete Funktionen, Felder einer Struktur   |
| allocate_memory,<br>free_queue, loopup_max | Funktionen die weniger oft verwendet werden   |
| TABLE_SIZE                                 | Konstanten  |

## Regeln zu Variablen

- Verwenden Sie für die Deklaration jeder Variablen eine **eigene Programmzeile**.
- Jede Variable muss initialisiert werden.
- Ihr Programm darf keine "magischen Zahlen" enthalten, verwenden Sie stattdessen Konstanten, die Sie entweder mit `#define ...` oder mit dem Schlüsselwort `const...` deklarieren.
- Globale Variablen müssen vermieden werden. Jede globale Variable muss ausführlich begründet werden. (Ausnahme: Konstanten)

## Formatierung der Programmcodes

- Schließen Sie die Code-Blöcke der Steueranweisungen if, else, while, for und do immer in geschweifte Klammern {} ein, auch wenn der Code-Block nur eine Zeile enthält.
  - Es ist sinnvoll, wenn nach der öffnenden geschweiften Klammer „{“ höchstens noch ein Kommentar folgen, aber kein ausführbarer Code mehr.  
Eine geschlossene geschweifte Klammer „}“ steht stets alleine in einer Zeile.
- Die maximale Länge einer Programmzeile beträgt 80 Zeichen.
- Halten Sie Sichtformatierungen ein.
- Sorgen Sie bei arithmetischen und logischen Ausdrücken durch erzwungene Typumwandlung dafür, dass alle Operanden des Ausdrucks denselben Datentyp haben.

## Formatierung der Programmcodes (Fortsetzung)

- Garantieren Sie bei arithmetischen und logischen Ausdrücken durch Klammerung, dass sie leicht und eindeutig lesbar sind.
- Verwenden Sie keine Zuweisungen oder Inkrement- bzw. Dekrementausdrücke als Operanden in Ausdrücken oder als Argumente von Funktionen.
  - Folgendes ist also unzulässig:
    - `zahl = (wert++) * 7; /* Inkrement als Operand */`
    - `zahl = (wert = 3) + offset; /* Zuweisung als Operand */`
- `goto` ist nicht zulässig.



## Geschachtelte Blöcke

- Rücken Sie geschachtelte Blöcke jeweils um 4 Blanks oder einen Tabulator ein.
- Verwenden Sie einen einheitlichen Stil.

| Beispiel   | alternativer Stil  |
|--|--|
| <pre>while (i &gt; 0) {     f(x);     i = i - 3; }</pre> | <pre>while (i &gt; 0) {     f(x);     i = i - 3; }</pre> |

## White spaces are for free – use them!

Leerzeichen und Zeilenumbrüche erhöhen die Lesbarkeit von Ausdrücken.

### Beispiel:

```
if ((a==b) &&(OK || (v>w)) || erg) {  
    ...  
}
```

} **schlecht  
formatiert**

```
if ( (a == b) && ( OK || ( v > w ) )  
    || erg  
    )  
{  
    ...  
}
```

} **gut  
formatiert**

## Brackets are for free – use them!

Vermeiden Sie durch Klammern Zweifel bei der Auswertungsreihenfolge innerhalb eines Ausdrucks.

### Beispiel:

|                                    |   |                                |
|------------------------------------|---|--------------------------------|
| <code>x = a + b &lt;&lt; 1;</code> | } | <b>schlecht<br/>formatiert</b> |
|------------------------------------|---|--------------------------------|

|                                      |   |                           |
|--------------------------------------|---|---------------------------|
| <code>x = (a + b) &lt;&lt; 1;</code> | } | <b>gut<br/>formatiert</b> |
|--------------------------------------|---|---------------------------|

Haben Sie gewusst, dass die Ausdrücke äquivalent sind?

## Geschachtelte if Anweisungen

**C Syntax:** Geschachtelte if Anweisungen in C: Ein else Teil gehört immer zur nächsten vorhergehenden if Anweisung, die noch keinen else Teil hat.

**Regel:** Klammern Sie stets alle geschachtelten if Anweisungen, so dass die Zuordnung der else Teile daraus hervorgeht.

**Beispiel:**

```
if (x < y)
    if (z != 0) x = z;
else
    x = y;
```

schlecht  
formatiert

```
if (x < y) {
    if (z != 0) {
        x = z;
    }
} else {
    x = y;
}
```

*Diese Codestücke  
sind nicht äquivalent.*

gut  
formatiert

Haben Sie gewusst, dass die gelb unterlegten Code Sequenzen äquivalent sind?

```
if (x < y) {
    if (z != 0) {
        x = z;
    } else {
        x = y;
    }
}
```

## Das NULL Statement

➤ Ein NULL Statement ist ein Statement, was „nichts tut“.

➤ ; ist ein NULL Statement in C.

➤ **Typischer Fehler:**

```
while (i > 0) ;  
{  
    ... block zu dem While Statement  
}
```

Das NULL  
Statement bildet  
den Rumpf des  
while Statements.

➤ **Regeln:**

- Ein NULL Statement steht stets zusammen mit einem Kommentar in einer eigenen Zeile.
- Schreiben Sie niemals ein Semikolon direkt hinter ein if, for, while ... Statement (Ausnahme do-while Schleifen).

## So schreibt man Kommentare

**Regel:** Kommentieren, aber nicht „über-kommentieren“.

- Diese Regel besagt, dass Kommentare genau so umfangreich sein müssen, dass man das Programm versteht.

### **Block-Kommentare:**

- Sie erstrecken sich über eine oder mehrere Zeilen.
- Sie kommentieren i.a. ein ganzes Codestück.
- Diese stehen auf der selben Einrücktiefe wie das Codestück, das sie beschreiben.

### **Inline-Kommentare:**

- Sie stehen am Ende einer Codezeile.
- Sie müssen kurz und prägnant sein.
- Inline-Kommentare, die den Code duplizieren, sind nutzlos.

## Fehlervermeidung bei Makros

Bezeichner von Makros bestehen nur aus Großbuchstaben und “\_”. So wird ihre besondere Behandlung durch den Präprozessor hervorgehoben.

**Regel:** Klammern Sie stets die Argumente eines Makros.

Klammern Sie stets den Ausdruck, den das Makro beschreibt. Anweisungen werden in einem Block zusammengefasst.

**Beispiel:**

```
/* schlechter, fehleranfälliger Stil */
#define MY_MULT(x,y) \
    x * y

/* guter Stil */
#define MY_MULT1(x,y) \
    ((x) * (y))

int i = 3 + MY_MULT (3 + 4, 4 + 1);    /* i = 23 */
int j = 3 + MY_MULT1 (3 + 4, 4 + 1);  /* j = 38 */
```

## Fehlervermeidung bei Makros (Fortsetzung)

**Regel:** Zwischen dem Makro Namen und der Argumentliste darf kein Blank stehen.

**Beispiel:**

*Dieses Blank ist falsch.*

```
#define MY_MULT1 (x,y)((x) * (y))
```

```
int j = 3 + MY_MULT1(3 + 4, 4 + 1);
```

Der Präprozessor erzeugt den Code

```
int j = 3 + (x,y)((x) * (y))(3 + 4, 4 + 1);
```

Aufgrund dieses Codes erzeugt der Compiler eine Fehlermeldung.



## Fehlervermeidung bei Makros (Fortsetzung)

**Regel:** Die aktuellen Parameter von Makros müssen seiteneffektfrei sein.

**Beispiel:**

```
#define DOUBLE(x) ((x) + (x))
```

```
j = DOUBLE(i++);
```

Der Präprozessor erzeugt den Code

```
int j = ((i++) + (i++));
```

Potentielles Problem: i wird im 2 erhöht.

## Wer Zeiger initialisiert, der findet Fehler schneller!

**Regel:** Initialisieren Sie jeden Zeiger mit NULL oder der Adresse eines Speicherplatzes des entsprechenden Typs.

**Beispiel:**

```
int *i;  
*i = 22;
```

Es wird nicht in jedem Fall erkannt, dass der Zeiger i keine gültige Adresse enthält.

```
int *i = NULL;  
*i = 22;
```

Es wird stets erkannt, dass der Zeiger i die ungültige Adresse NULL enthält.

## Fehlerquellen bei Zuweisungen

### Regeln:

- Vermeiden Sie Zuweisungen in Ausdrücken.

### Beispiele:

```
x += (x == y); // schlechter Stil, schwer lesbar
```

```
if (x == y)    // guter Stil, leicht lesbar
    x++;
```

```
if (x = y < z) { // schlechter Stil
    ...
}
```

```
x = y;
if (x < z) {    // guter Stil
    ...
}
```

Die Codesequenzen sind nicht äquivalent, denn < bindet stärker als =.

## Seiteneffekte und Zuweisungen

**Regel:** Verwenden Sie keine Zuweisungen oder Inkrement- bzw. Dekrementausdrücke als Operanden in Ausdrücken oder als Argumente von Funktionen. Dies gilt auch für Funktionen mit Seiteneffekt.

### Beispiele:

```
xs[i++] = i;    /* schlechter Stil */
```

```
*p++ = *p - x; /*schlechter Stil */
```

```
*p = *p - x;    /* guter Stil */  
p++;
```

```
rotate(*p++, *p++);    /* schlechter Stil */
```

```
rotate(*p, *(p + 1)); /* guter Stil */  
p += 2;
```

## Zusammenfassung

Es wurden grundlegende C- Coding Style Regeln vorgestellt.

Betrachten Sie diese Regeln als Startbasis.

Für alle Modifikationen/Erweiterungen gilt:

- Beachten Sie die drei goldenen Regeln aus der Einleitung
  - **Keep it simple.**
  - **Don't be clever.**
  - **Be explicit – write your program for people.**

Beachten Sie stets folgende Regel:

- **Write your code the same way as the code around it.**