Coding Style für C

Robert Heß

18. April 2017

Inhaltsverzeichnis

1 Einleitung

1.1 Was bedeutet *Coding Style*?

Neben technisch korrektem Programmieren ist ein guter Programmierstil nicht nur hilfreich, sondern bei großen Programmen von elementarer Bedeutung. Der *Coding Style* beschreibt einige Richtlinien, die es anderen Programmierern, aber auch dem Autor selbst erleichtert, den Quellcode schnell zu verstehen.

Coding Style ist bis zu einem gewissen Grad etwas subjektives, und jeder hat seine individuellen Vorlieben. Innerhalb einer Gruppe hilft aber ein einheitlicher Coding Style, dass alle Beteiligten eine einheitliche Sprache sprechen und besser miteinander über Ihre Software kommunizieren können.

1.2 Wozu *Coding Style*?

Einige Vorteile sind [1]:

- Einheitlicher Programmierstil erleichtert die Einarbeitung
- Die Programme sind einfach zu lesen und zu verstehen
- Die Programme lassen sich leichter in eine andere Umgebung portieren
- Einige typische Fehler werden vermieden
- Mehrere Programmierer können leichter an einem Programm arbeiten

Kurzum, ein Programm soll so gestaltet werden, dass sich ein Außenstehender schnell darin zurecht findet.

1.3 Allgemeines

Über die Jahre hat sich in vielen Bereichen eine gängige Praxis entwickelt. Auf der anderen Seite definiert jede Firma ihren eigenen Coding Style. Für unsere Vorlesung soll dies mit diesem Dokument erfolgen.

Große Teile dieses Dokuments sind den Richtlinien für den Linux-Kern entnommen [2].

2 Aufbau einer Datei

Eine Quellcodedatei ist wie folgt aufgebaut:

1. Kommentarkopf, siehe Abschnitt 7
2. Einbinden von Header-Dateien
3. Definition von Datentypen (*struct*, *typedef* etc.) und Makros
4. Deklaration von Funktionen
5. Definition von globalen Variablen
6. Hauptprogramm *main()*, wenn vorhanden
7. Definition (Implementierung) der Funktionen

Eine Headerdatei hat eine ähnliche Struktur:

1. Kommentarkopf, siehe Abschnitt 7
2. Einbinden von Header-Dateien
3. Definition von Datentypen (*struct*, *typedef* etc.) und Makros
4. Deklaration von Funktionen

3 Bezeichner

- Erlaubte Zeichen: Buchstaben *A-Z*, *a-z*, Ziffern *0-9* und Unterstrich _
- Daraus folgt, keine Sonderzeichen wie die deutschen Umlaute oder das scharfe S verwendet werden sollen
- Bezeichner für Makros werden nur mit Großbuchstaben und Unterstrich gebildet
- Namen von Variablen, Funktionen und Parametern mit Kleinbuchstaben
- Zusammengesetzte Bezeichner werden durch Unterstrich (sog. *snake_case*) oder Großbuchstaben (sog. *lowerCamelCase*) angedeutet, z.B:
  spieler_eins oder spielerEins
  Im gesamten Programm soll nur eine Art verwendet werden
- Verwenden Sie aussagekräftige Bezeichner
- Vermeiden Sie optisch ähnliche Namen. Beispiel: index1 und indexl
- Integrieren Sie nicht den Datentyp in den Namen (Ungarische Notation), also nicht:

```
int int_zahl;     // so nicht!
```

4 Einrückung/Ausrichtung

In C hat das Einrücken keine funktionale Bedeutung, aber durch sinnvolles Einrücken wird die Struktur des Programms verdeutlicht.

- Definieren Sie eine Einrückweite von 4 bis 8 Zeichen. Eine Einrückweite von 4 Zeichen hat sich bewährt.
- Verwenden Sie für das Einrücken Tabulatoren, so dass ggf. später die Einrücktiefe angepasst werden kann.
- Pro Verschachtlung eine Einrückung, z.B.:

```
int main()
{
    int i;
    int fact[10];

    for(i=0; i<10; i++) {
        if(i<2)
            fact[i] = 1;
        else
            fact[i] = i*fact[i-1];
    }
}
```

- Platzieren Sie jeden Befehl in eine eigene Zeile.
  Ausnahme: befindet sich in einer Schleife oder Verzweigung nur ein Befehl, so kann dieser dahinter platziert werden, z.B.:

```
if(i<2) fact[i] = 1;
else fact[i] = i*fact[i-1];
```

- Vermeiden Sie komplizierte, schwer nachvollziehbare Ausdrücke
- Vermeiden Sie Leerzeichen und Tabulatoren am Ende einer Zeile
- Vermeiden Sie Zeilen breiter als 80 Zeichen inkl. Kommentare und Leerzeichen
- Erhöhen Sie die Übersicht Ihres Quellcodes durch Verwendung von Leerzeilen

5 Geschweifte Klammern und Leerzeichen

- Außer bei Funktionen wird die öffnende Klammer hinter den entsprechenden Befehl gesetzt:

```
if(a<b) {
    /* ... */
}
```

- Bei einer Funktion wird die Klammer unter den Funktionskopf gesetzt:

```
void Funktion(int x)
{
    printf("Wert von x: %d\n",x);
}
```

- Außer beim *while* der *do*-Schleife und beim *else* befindet sich die schließende Klammer immer in einer separaten Zeile, z.B.:

```
if(a!=b) {
    /* ... */
} else {
    /* ... */
}
```

- Klammen um einzelne Befehle werden vermieden:

```
if(a!=b)
    printf("a ungleich b\n");
else
    printf("a gleich b\n");
```

- Bei der Definition von Variablen wird ein Leerzeichen direkt hinter den elementaren Datentyp eingefügt, z.B.:

```
int *a;    // gutes Beispiel
int* b;    // schlechtes Beispiel
```

- Um binäre und ternäre Operatoren werden Leerzeichen eingefügt. Ausnahmen: -> und ., Beispiel:

```
x = point.x + data->x;
```

- Hinter unären Operatoren (+, -, !) wird kein Leerzeichen eingefügt
- Zwischen Inkrement-/Dekrementoperatoren und den dazugehörigen Variablen wird kein Leerzeichen eingefügt

6 Funktionen

- Funktionen werden zu Beginn einer Quellcodedatei oder in einer passenden Headerdatei deklariert
- Funktionen werden in der Quellcodedatei definiert (implementiert)
- Falls vorhanden werden die Funktionen unterhalb von *main()* definiert
- Innerhalb einer Funktion werden alle lokalen Variablen zu Beginn definiert gefolgt von einer Leerzeile
- Zwischen Funktionen wird eine Leerzeile eingefügt
- Funktionen werden durch eine Kommentarzeile optisch von einander getrennt, siehe Abbildung 1

7 Kommentare

- Es können beide Arten von Kommentaren verwendet werden: /*...*/ und //
- Jede Datei hat einen Dateikopf mit den Fünf Elementen: Dateiname, Autor, Version, Datum und Beschreibung
- Alle Variablen werden rechts neben der Definition beschrieben, siehe Abbildung 1
- Vermeiden Sie Kommentare ohne Aussage, z.B.:

```
// bad example:
int count; // int-variable 'count'
```

- Logische Blöcke werden oberhalb mit einem Kommentar versehen

8 Verschiedenes

- Vermeiden Sie globale Variablen (in unserem Kurs ausschließlich)
- Für einen übersichtlichen Programmablauf vermeiden Sie *goto*, *continue* und *break*. Letzteres wird nur im Kontext von *switch* verwendet.
- Verwenden Sie für Namen und Kommentare einheitlich eine Sprache. Zur Auswahl stehen in diesem Kurs deutsch und englisch.

Literatur

[1] M Henricson and E Nyquist: *Programming in C++, Rules and Recommendations*, Ellemtel Telecommunication Systems Laboratories (1992), www.literateprogramming.com/ellemtel.pdf, abgerufen am 15.3.2017

[2] *Linux kernel coding style*, www.01.org/linuxgraphics/gfx-docs/drm/process/coding-style.html, abgerufen am 17.4.2017

```c
// filename: factorial.c
// version: 1.0
// date: April 3rd 2017
// author: Robert Hess
// description: Funtion to evaluate the factorial

#include <stdio.h>

double factorial(unsigned n);

//=============================================================
int main()
{
    unsigned x;  // index to evalaute the factorial

    // loop to evaluate factorials
    for (x = 0; x <= 170; x++)
        printf("%2u! = %lg\n", x, factorial(x));

    return 0;
}

//=============================================================
double factorial(     // returns the factorial of n
    unsigned n)       // argument for calculation
{
    double fact = 1;  // result of calculation

    // evalaute factorial
    while (n > 1) {
        fact *= n;
        n--;
    }

    // return factorial
    return fact;
}
```

Abbildung 1: Beispiel für einen Quellcode.