C-Quelltextstil

Bearbeitet von Andreas Westfeld (Version vom 27. Januar 2020) in Anlehnung an den *Linux kernel coding style* in der Version vom 13. Juli 2007 m. w. N.

§ 1 Einrücktiefe

(1) Anfang und Ende eines Blocks sollen klar sichtbar sein. Tabulatoren sind 8 Zeichenspalten breit, deshalb werden Blöcke 8 Zeichenspalten tief eingerückt. Zum Einrücken werden ausschließlich Tabulatorzeichen verwendet, keine Leerzeichen. Am Ende einer Zeile stehen weder Leerzeichen noch Tabulatoren.

(2) Marken, zu denen mit `goto` gesprungen werden kann, beginnen in der ersten Spalte.

```
int include_name(const char *name)
{
        int retval;

        lock_area();
        retval = prepare(name);
        if (retval < 0) {
                fprintf(stderr, "Unable to prepare %s\n", name);
                goto error;
        }
        push_string(name);
error:
        unlock_area();
        return retval;
}
```

(3) In `switch`-Anweisungen beginnen das Schlüsselwort `switch` und die zugehörigen `case`-Marken in derselben Spalte.

```
        switch (suffix) {
        case 'G':
        case 'g':
                mem <<= 30;
                break;
        case 'M':
        case 'm':
                mem <<= 20;
                break;
        case 'K':
        case 'k':
                mem <<= 10;
                /* fall through */
        default:
                break;
        }
```

(4) Jede Anweisung gehört auf eine neue Zeile.

§ 2 Zeilenlänge

(1) Die Länge einer Zeile ist auf 80 Zeichenspalten beschränkt.

(2) Umbrüche sind zu vermeiden. Innerhalb von Anweisungen darf nur umgebrochen werden, soweit dies erforderlich ist, um die 80. Zeichenspalte nicht zu überschreiten. Umgebrochene Teile von Anweisungen werden deutlich nach rechts abgesetzt (grundsätzlich zwei Tabulatorzeichen mehr als das erste Zeichen der Anweisung).

(3) Absatz 2 ist auf die Kopfzeile einer Funktion mit einer langen Parameterliste entsprechend anzuwenden.

(4) Lange Zeichenketten werden ggf. in kürzere umgebrochen.

```
void fun(int a, int b, int c)
{
        if (condition)
                printk(KERN_WARNING "Warning this is a long printk with "
                                "3 parameters a: %u b: %u c: %u \n", a, b, c);
        else
                next_statement;
}
```

§ 3 Geschweifte Klammern

(1) Die öffnende geschweifte Klammer steht stets am Zeilenende.

```
        if (x is true) {
                we do y
        }
```

(2) Grundsätzlich steht die öffnende geschweifte Klammer auf derselben Zeile wie die Schlüsselwörter `if`, `switch`, `for`, `while` und `do`, zu welchen der Anweisungsblock gehört.

```
        switch (action) {
        case KOBJ_ADD:
                return "add";
        case KOBJ_REMOVE:
                return "remove";
        case KOBJ_CHANGE:
                return "change";
        default:
                return NULL;
        }
```

(3) Funktionskörper werden durch eine öffnende geschweifte Klammer auf separater Zeile eingeleitet. Die Klammer steht dabei unter der Spalte, in der der Funktionskopf beginnt.[1]

[1] Diese scheinbare Inkonsistenz wurde bereits durch Kernighan und Ritchie geübt.

```
int function(int x)
{
        body of function
}
```

(4) Die schließende geschweifte Klammer steht grundsätzlich allein auf einer ansonsten leeren Zeile, und zwar in der Spalte, in der das den Anweisungsblock begründende Schlüsselwort beginnt oder, sofern es kein solches gibt, in der Spalte, in der die öffnende geschweifte Klammer steht.

(5) Sofern die Anweisung nach der schließenden geschweiften Klammer noch nicht beendet ist, beispielsweise durch das Schlüsselwort `else` einer `if`-Anweisung oder das Schlüsselwort `while` einer `do`-Schleife, wird sie auf derselben Zeile fortgesetzt.

```
if (x == y) {
        ..
} else if (x > y) {
        ...
} else {
        ....
}

do {
        body of do-loop
} while (condition);
```

§ 4 Leerzeichen

(1) Nach einer öffnenden Klammer und vor einer schließenden Klammer darf kein Leerzeichen stehen.

(2) Auf die Schlüsselwörter `if`, `switch`, `case`, `for` und `while` folgt stets ein Leerzeichen. Wird im Zusammenhang mit `if`, `switch`, `do`, `for` oder `while` ein Block eingeleitet, so steht vor der öffnenden geschweiften Klammer ein Leerzeichen.

(3) Bei der Deklaration von Zeigern oder Funktionen, die einen Zeiger zurückgeben, steht zwischen Typ und `*` ein Leerzeichen. Zwischen `*` und Name steht kein Leerzeichen.

```
char *linux_banner;
unsigned long long memparse(char *ptr, char **retptr);
char *match_strdup(substring_t *s);
```

(4) Vor und nach binären Operatoren steht grundsätzlich ein Leerzeichen. Die Operatoren `->` und `.` stehen ohne Leerzeichen. Das Komma folgt ohne Leerzeichen und wird selbst von einem Leerzeichen gefolgt.

```
a->b, c.d, a + b, x == y
```

(5) Nach unären Operatoren folgt kein Leerzeichen.

```
~a, *b, !c, -d, ++x, (char *)p
```

(6) Vor unären Operatoren in Postfix-Notation steht kein Leerzeichen.

```
a++, b--
```

§ 5 Namen

Namen in C sollen keine Großbuchstaben enthalten. Mit `#define` erzeugte Präprozessorsymbole sollen vorzugsweise Großbuchstaben verwenden. Global sichtbare Namen sollen sprechende Bezeichnungen erhalten.

lokale Namen: `int i, j, tmp, *p;`
globale Namen: `int count_active_users(void);`

```
#define CONSTANT 0x12345
#define ARRAY_SIZE(x) (sizeof(x) / sizeof((x)[0]))
```

§ 6 Funktionen

(1) Funktionsdefinitionen sollen nicht länger als 48 Zeilen sein, es sei denn, ihre erforderliche maximale Einrücktiefe überschreitet nicht zwei Tabulatoren.

(2) Innerhalb einer Funktion sollen nicht mehr als 10 lokale Variablen definiert werden.

(3) Funktionsdefinitionen sollen durch eine Leerzeile voneinander getrennt sein.

(4) Funktionsprototypen sollen Parameternamen und deren Typen enthalten.

§ 7 Kommentare

(1) Der Quelltext soll keine Kommentare im C99-Stil „`//` ...“ enthalten, sondern sich am C89-Standard „`/*` ... `*/`“ orientieren.

(2) Mehrzeilige Kommentare sollen dem unten angegebenen Beispiel folgen. Die Sternchen dürfen abweichend von § 1 Absatz 1 zusätzlich mit einem Leerzeichen eingerückt werden.

```
/*
 * This is the preferred style for multi-line comments.
 * Please use it consistently.
 *
 * Description: A column of asterisks on the left side,
 * with beginning and ending almost-blank lines.
 */
```