Programmierstile in C

Von Sebastian Rothe

Inhaltsverzeichnis

Motivation – „wozu Code-Formatierung?“

- Selten wird eine Software nur von ihrem ursprünglichen Autor weiter gepflegt
- Gute Code-Formatierung verbessert die Lesbarkeit
- Leichtere, schnellere und flexiblere Erweiterung der Software
- Richtwert: 80% der Lebenszeit einer Software entfällt auf die Wartung

Was versteht man unter dem Begriff „Programmierstil“?

- Ein Satz von Regeln, nach denen der Programmierer den Code gestaltet
- Greift verschiedene Aspekte auf:
  - Quelltextformatierung
  - Namenskonventionen
  - Wiederverwendbarkeit/Wartung
  - Dokumentation des Codes
  - Robustheit durch Fehlerbehandlung
  - Modularität der Software

Und was ist „guter Programmierstil“?

- Guter Programmierstil ist immer subjektiv
- Es gibt keine feste Definition, was alles zu gutem Programmierstil zählt
- Verschiedene umstrittene Elemente, Beispiele:
  - Einrückungsstil
  - Zeilenlänge
  - Namenskonventionen
  - Left-hand- vs. Right-hand-comparisons
- Innerhalb eines Teams: Einigung auf bestimmte Konventionen
- Diese sollten dann auch konsequent verwendet werden

- Einige Konventionen werden bei sehr vielen Projekten eingehalten:
  - Nur eine Anweisung pro Zeile
  - Zeilenlänge auf 80 Zeichen beschränken
  - Gängige Namenskonventionen (dazu später mehr)
  - Vorgaben zur Dokumentation

Elemente des Einrückungsstils

- Positionierung von Klammern
- Tiefe der Einrückung
- Verwendung von Leerzeichen/Tabulatorzeichen

- Unterschiedliche Bezeichnungen für den (fast) gleichen Stil
- Meistens abhängig von der verwendeten Programmiersprache

K&R / 1TBS / Kernel / Unix / „West Coast“ / Stroustrup / Java / Sun

- K&R: benannt nach *Kernighan* und *Ritchie*, den Entwicklern von C
- Öffnende Klammern am Ende eines Ausdrucks
- Schließende auf dem gleichen Level wie der Ausdruck
- Je nach Sprache zwischen 3 und 8 Leerzeichen pro Einrückungsschritt

```
1  int f(int x, int y, int z)
2  {
3          if (x < foo(y, z)) {
4                  haha = bar[4] + 5;
5          } else {
6                  while (z) {
7                          haha += foo(z, z);
8                          z--;
9                  }
10                 return ++x + bar();
11         }
12 }
```

Quelle: [1]; *Variation des K&R, Funktionsdefinitionen nach Allman*

```
1  int f(int x, int y, int z) {
2      if (x < foo(y, z)) {
3          haha = bar[4] + 5;
4      } else {
5          while (z) {
6              haha += foo(z, z);
7              z--;
8          }
9          return ++x + bar();
10     }
11 }
```

Quelle: [2]; *Variation des K&R: Java / Sun*

Allman / BSD / „East Coast“ / Horstmann

- Bessere Übersicht der umschließenden Klammern
- Aber: pro Block zusätzliche Zeilen für Klammerung
- Wieder die Frage: guter Programmierstil?

```
 1 int f(int x, int y, int z)
 2 {
 3     if (x < foo(y, z))
 4     {
 5         haha = bar[4] + 5;
 6     }
 7     else
 8     {
 9         while (z)
10         {
11             haha += foo(z, z);
12             z--;
13         }
14         return ++x + bar();
15     }
16 }
```

Quelle: [3]; *Code-Beispiel nach Allman*

```
 1 int f(int x, int y, int z)
 2 {   if (x < foo(y, z))
 3     {
 4         haha = bar[4] + 5;
 5     }
 6     else
 7     {
 8         while (z)
 9         {
10             haha += foo(z, z);
11             z--;
12         }
13         return ++x + bar();
14     }
15 }
```

Quelle: [4]; *Code-Beispiel nach Horstmann*

GNU-Stil

- Vor allem, aber nicht überwiegend, in GNU-Projekten genutzt
- Argument: bessere Lesbarkeit durch Einrückung der geschweiften Klammern
- Gegenargument: Verschlechterung der Lesbarkeit durch verwirrende Einrückung

```
 1 int f (int x, int y, int z)
 2 {
 3     if (x < foo (y, z))
 4       haha = bar[4] + 5;
 5     else
 6       {
 7         while (z)
 8           {
 9             haha += foo (z, z);
10             z--;
11           }
12         return ++x + bar ();
13       }
14 }
```

Quelle: [5]; *Code-Beispiel im GNU-Stil*

- Es gibt unzählige Einrückungsstile
- Nur selten Einschränkung durch vorgegebene Syntax
- Gute Editoren unterstützen den Programmierer oftmals

> *“The position of braces is less important, although people hold passionate beliefs. We have chosen one of several popular styles. Pick a style that suits you, then use it consistently.”*
>
> Zitat von Brian Kernighan und Dennis Ritchie aus ihrem Werk *The C Programming Language*

Vertikale Anordnung

```
1 int einArray[5] = {1, 2, 3, 4, 5};
2 int nochEinArray[5] = {6, 7, 8, 9, 0};
3
4 int einWert = 0;
5 int nochEinWert = 1;
6 int nochEinAndererWert = 2;
```

Quelle: [6]

```
1 int einArray[5]     = {1, 2, 3, 4, 5};
2 int nochEinArray[5] = {6, 7, 8, 9, 0};
3
4 int einWert             = 0;
5 int nochEinWert         = 1;
6 int nochEinAndererWert  = 2;
```

Quelle: [7]

- Besserer Bezug der Variablen zueinander
- Zugewiesene Werte lassen sich besser vergleichen/überprüfen
- Alternative Anordnung (z.B. in PHP):

```
1 <?php
2             $einWert = 0;
3     $einAndererWert = 1;
4 $nochEinAndererWert = 2;
5 ?>
```

Quelle: [8]

- Aber: Aufwändigere Code-Formatierung, ggf. Probleme bei Änderungen

Leerzeichen

```
1 int i;
2 for(i=42;i>0;--i){
3   printf("%d \n",i);
4 }
```

oder

```
1 int i;
2 for(i=42; i>0; --i){
3   printf("%d \n", i);
4 }
```

oder

```
1 int i;
2 for(i = 42; i > 0; --i){
3   printf ("%d \n", i);
4 }
```

- „Whitespace“ dient in vielen Sprachen der Lesbarkeit
- Beispiele bedeuten syntaktisch das Gleiche
- Jeder hat einen bevorzugten Stil
- Trotzdem sollte man sich im Team auf einen Stil einigen

Tabs

```
1 int i;
2 const int x;
3 double j;
4 char c;
5 char s[];
```

oder

```
1 int        i;
2 const int  x;
3 double     j;
4 char       c;
5 char       s[];
```

Leerzeichen *oder* Tabulator?

- Greift auf den Aspekt Einrückungsstile zurück
- Sollten Einrückungen durch Leerzeichen oder Tabs vorgenommen werden?
- Beide Methoden haben ihre Vorteile:
  - Leerzeichen: konstante Darstellung unabhängig von Anzeigeoptionen
  - Tabulator: jeder Programmierer kann die Einrückungstiefe selbst wählen
- Auch hier gilt: Varianten sollten nicht gemischt werden

Left-hand- vs. Right-hand-comparisons

- Kleines Codebeispiel (Perl) zur Verdeutlichung:

```
# Vergleich:
if($einWert == 42) {...} # Right-hand
if(42 == $einWert) {...} # Left-hand
```

Quelle: [9]

```
# Achtung! Zuweisung:
if($einWert = 42) {...} # Right-hand
if(42 = $einWert) {...} # Left-hand
```

Quelle: [10]

- Left-hand-comparison: der zu vergleichende Ausdruck steht links
- Right-hand-comparison ist dementsprechend das Gegenteil
- Right-hand-comparison wird für gewöhnlich genutzt
- Rechts: Bug bei der Right-hand-comparison kann übersehen werden

- Lesbarkeit (/Gewohnheit) oder Funktionalität?

- Kommentare sollten ausgiebig genutzt werden
- Unerlässlich für Team-Projekte
- Aber auch bei Kommentaren gibt es „Richtlinien“:
  - Der Code soll nicht erklärt oder wiederholt werden
  - Die Absicht soll deutlich werden
  - Kommentare können als Platzhalter verwendet werden
  - Praktisch zum Debuggen („Auskommentieren“)

```
1 /*
2 *  Dies ist ein Block-Kommentar,
3 *  der sich über mehrere Zeilen
4 *  hinweg verteilen kann
5 */
```

```
1 // Es gibt auch einzeilige Kommentare
2
3 // Diese können aber vereinzelt Probleme bereiten
```

- Einzeilige Kommentare zählen erst ab 1999 zum C-Standard

- Oft verwendet: *Ungarische Notation*

- Typ der Variable durch Präfix des Bezeichners erkennbar:

```
1 int iSumme;
2
3 long lGrosseZahl;
4
5 char chZeichen;
6
7 char stString[];
8
9 int const iUMSATZSTEUER;
```

- Typische Namenskonventionen für C:
  - Konstanten werden in Großbuchstaben deklariert
  - Funktionen, Typendefinitionen und Variablen: Kleinbuchstaben
  - Bezeichner mit führendem Unterstrich sollten nicht verwendet werden
  - Namen vermeiden, die sich in nur einem Buchstaben unterscheiden
  - Namen vermeiden, die ähnlich aussehen (kleines L, großes I, 1)

Was ist Glib?

- GLib ist eine Bibliothek für C
- Enthält verschiedene, sonst nur schwer zu realisierende Funktionen
- Beinhaltet eine Vielzahl komplexer Funktionen:
  - Makros
  - Basistypen
  - Typumwandlung
  - Timer
  - Datenstrukturen
  - Threads
  - Diverse String-Verarbeitungsmöglichkeiten

GLib als Beispiel

```
170 /* Arch specific stuff for speed
171  */
172 #if defined (__GNUC__) && (__GNUC__ >= 2) && defined (__OPTIMIZE__)
173 #  if defined (__i386__)
174 #    define GUINT16_SWAP_LE_BE_IA32(val) \
175        (__extension__                     \
176   ({ register guint16 __v, __x = ((guint16) (val)); \
177      if (__builtin_constant_p (__x))      \
178        __v = GUINT16_SWAP_LE_BE_CONSTANT (__x);   \
179      else                 \
180        __asm__ ("rorw $8, %w0"        \
181           : "=r" (__v)          \
182           : "0" (__x)        \
183           : "cc");           \
184       __v; }))
185 #    if !defined (__i486__) && !defined (__i586__) \
186   && !defined (__pentium__) && !defined (__i686__) \
187   && !defined (__pentiumpro__) && !defined (__pentium4__)
188 #       define GUINT32_SWAP_LE_BE_IA32(val) \
189     (__extension__                 \
190      ({ register guint32 __v, __x = ((guint32) (val));  \
191         if (__builtin_constant_p (__x))      \
192     __v = GUINT32_SWAP_LE_BE_CONSTANT (__x);  \
193         else                \
194     __asm__ ("rorw $8, %w0\n\t"      \
195        "rorl $16, %0\n\t"      \
196        "rorw $8, %w0"        \
197        : "=r" (__v)        \
198        : "0" (__x)         \
199        : "cc");        \
200         __v; }))
201 #    else /* 486 and higher has bswap */
202 #       define GUINT32_SWAP_LE_BE_IA32(val) \
203     (__extension__                 \
204      ({ register guint32 __v, __x = ((guint32) (val));  \
205         if (__builtin_constant_p (__x))      \
206     __v = GUINT32_SWAP_LE_BE_CONSTANT (__x);  \
207         else                \
208     __asm__ ("bswap %0"        \
209        : "=r" (__v)        \
210        : "0" (__x));         \
211         __v; }))
212 #    endif /* processor specific 32-bit stuff */
```

Quelle: [11]

- Verschiedene Aspekte der angesprochenen Konventionen erkennbar
- Schwerpunkt: vertikale Anordnung
- Kommentare für große if-Anweisungen

GLib als Beispiel

```
0        10        20        30        40        50        60        70        80        90        100
27 #if defined(G_DISABLE_SINGLE_INCLUDES) && !defined (__GLIB_H_INSIDE__) && !defined (GLIB_COMPILATION)
28 #error "Only <glib.h> can be included directly."
29 #endif
30
31 #ifndef __G_TIMER_H__
32 #define __G_TIMER_H__
33
34 #include <glib/gtypes.h>
35
36 G_BEGIN_DECLS
37
38 /* Timer
39  */
40
41 /* microseconds per second */
42 typedef struct _GTimer    GTimer;
43
44 #define G_USEC_PER_SEC 1000000
45
46 GTimer*  g_timer_new           (void);
47 void   g_timer_destroy         (GTimer      *timer);
48 void   g_timer_start           (GTimer      *timer);
49 void   g_timer_stop          (GTimer      *timer);
50 void   g_timer_reset           (GTimer      *timer);
51 void   g_timer_continue        (GTimer      *timer);
52 gdouble  g_timer_elapsed         (GTimer      *timer,
53           gulong      *microseconds);
54
55 void     g_usleep                (gulong       microseconds);
56
57 void     g_time_val_add          (GTimeVal    *time_,
58                                   glong        microseconds);
59 gboolean g_time_val_from_iso8601 (const gchar *iso_date,
60           GTimeVal    *time_);
61 gchar*   g_time_val_to_iso8601   (GTimeVal    *time_) G_GNUC_MALLOC;
62
63 G_END_DECLS
64
65 #endif /* __G_TIMER_H__ */
66
```

Quelle: [12]

- Hierbei kann zwischen verschiedenen Möglichkeiten unterschieden werden:
  - Tools zur Code-Analyse
  - Tools zur Code-Formatierung
  - Unterstützung des Programmierers durch den Editor
- Beispiel für C-Code-Formatierung: *Indent*

```
indent -nbad -bap -nbc -bbo -hnl -br -brs -c33 -cd33 -ncdb -ce -ci4
       -cli0 -d0 -di1 -nfc1 -i8 -ip0 -l80 -lp -npcs -nprs -npsl -sai
       -saf -saw -ncs -nsc -sob -nfca -cp33 -ss -ts8 -il1
       code.c -o code_formatiert.c
```

Quelle: [13]; *Indent, äquivalente Formatierung für Linux-Stil*

-bap: blank line after procedure body
-br: braces on if-line
-brs: braces on struct declaration line
-saf: space after for
usw…

Einige für C-Programmierung geeignete Editoren

- Bloodshed Dev-C++ (auch für C geeignet)
- Notepad++
- PSPad
- Eclipse IDE for C/C++ Developers
- Microsoft Visual Studio
- NetBeans IDE

Zusammenfassung

- Es gibt unterschiedlichste Stile
- Jeder hat seine eigene Auffassung von gutem Programmierstil
- Wichtig ist, diesen für sich persönlich auch konsequent zu nutzen
- Im Team sollte ein einheitlicher Stil genutzt werden
- Programmierstile stellen keine Regeln, sondern eher Richtlinien auf
- Diese können gut begründet auch in Einzelfällen umgangen werden

Quellen

- *http://de.wikipedia.org/wiki/Programmierstandard*
- *http://en.wikipedia.org/wiki/Indent_style*
- *http://google-styleguide.googlecode.com/svn/trunk/cppguide.xml*
- *http://en.wikipedia.org/wiki/Programming_style*
- *http://en.wikipedia.org/wiki/Coding_conventions*
- *http://en.wikipedia.org/wiki/Naming_convention_(programming)*
- *http://golang.org/cmd/gofmt/*
- *http://de.wikipedia.org/wiki/C_%28Programmiersprache%29*
- *http://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis#C*
- *http://www.dmoz.org/Computers/Programming/Development_Tools/Source_Code_Formatters/*
- *http://www.gnu.org/software/indent/*
- *http://stackoverflow.com/questions/411249/coding-style-checker-for-c*
- *http://wr.informatik.uni-hamburg.de/teaching/sommersemester_2011/c-programmierung*
- *http://wr.informatik.uni-hamburg.de/teaching/organisatorische_hinweise#proseminare*
- *http://www.inf.udec.cl/~leo/cstyguid.pdf*
- *http://www.psgd.org/paul/docs/cstyle/cstyle11.htm*
- *http://pronix.linuxdelta.de/C/standard_C/c_programmierung_5.shtml*

- Codebeispiele [1] – [5]: *http://de.wikipedia.org/wiki/Einrückungsstil*
- Codebeispiele [6] – [8]: *http://en.wikipedia.org/wiki/Programming_style*
- Codebeispiel [9] + [10]: *http://en.wikipedia.org/wiki/Coding_conventions#Left-hand_comparisons*
- Codebeispiel [11]: GLib Version 2.26.1 (Windows), *\include\glib-2.0\gtimer.h*
- Codebeispiel [12]: GLib Version 2.26.1 (Windows), *\include\glib-2.0\gtypes.h*
- Codebeispiel [13]: *http://www.gnu.org/software/indent/manual/indent.html#SEC4*
- Alle Codebeispiele wurden mit *PSPad* noch einmal überarbeitet