Kommentierung in C

Von Sebastian Rothe

Inhaltsverzeichnis

- Definition: Richtlinien, nach denen der Code gestaltet werden soll
- Der Begriff „Programmierstil“ umfasst i.d.R. verschiedene Aspekte:
  - Quelltextformatierung
  - Namenskonvention
  - Wiederverwerdbarkeit/Wartung
  - Modularität der Software
  - Robustheit durch Fehlerbehandlung
  - Und eben auch die Dokumentation des Codes

- Kommentierung wird oftmals unterschätzt und daher vernachlässigt, aber:
- Eine Software wird selten nur von ihrem Autor weitergepflegt
- Ausreichende und sinnvolle Kommentierung verbessert den „Lesefluss“
- Leichtere, schnellere und flexiblere Erweiterung der Software
- Ungefähr 80% der Lebenszeit einer Software entfällt auf die Wartung

„Effiziente Programmierung in C“ - der Bezug zum Seminartitel?

- Der Effizienzgewinn liegt hier nicht in der Performance der Software
- Die Effizienz liegt in der Erstellung des Codes
- Außerdem kann vorhandene Software effizienter gewartet werden
- Kommentierung kann präventiv für effizienteren Code sorgen

Was ist ein unsinniger Kommentar?

- Kommentare sollen den Quelltext nicht noch einmal wiederholen
- Einfache Zusammenhänge benötigen keine Erklärung

Was ist ein sinnvoller Kommentar?

- Ein Kommentar soll umfangreiche Codepassagen zusammenfassen
- Er soll komplizierte Anweisungen einfach darstellen
- Er soll helfen, den Code gut zu strukturieren

So sollte man es nicht machen:

```
 8  int functionOne(int arg1, int arg2, int arg3){
 9      int result; //result var
10      int sum;    //sum var
11
12      sum = /*arg1 from function call*/ arg1 + arg2; //sum of arg1 and arg2
13      int produkt = doProduct(arg3, sum); //product call for arg3 & sum
14      //DEPRECATED!!!!
15      //doSomethingElseWithSum(sum);
16      //TODO find a better idea of what to do with sum
17
18      if(sum>arg3){
19          //does something with sum
20          doSomethingWithSum(sum);
21      } //if(sum>arg3)
22      else if(arg3>sum){
23          //doSomethingWithArg(arg3);
24          doSomethingElseWithArg(arg3);
25      } //else if(arg3>sum)
26      else doNothing();
27      //do nothing
28      return 0; //return 0 because we can
29  }
```

Quelle: [1]

So sollte man es aber auch nicht machen:

```
int main(int argc, char **argv){

    bmpfile_t *bmp;
    int iProcess = 1;
    int iPixelSize = 1;
    int iSizeX = 0, iSizeY = 0;
    char *cDataDirName = "./data/";

    DIR *dirHandle;
    struct dirent *dirEntry;
    dirHandle = opendir(cDataDirName);
    if( NULL != dirHandle ){
    int iFileCounter = 0;
    while( NULL != ( dirEntry = readdir( dirHandle ) ) ){
        if( dirEntry->d_name[0] != '.' ){
    char cFilename[200] = "./data/";
    strcat(cFilename, dirEntry->d_name);
    FILE *datei;
    char *cText;
    int iRowCounter = 0, iActualAdds = 0;
    datei = fopen(cFilename, "r");
    if(datei != NULL){
        int iFirstRow = 1;
        int iFirstValues = 0;
        int iNewRow = 0;
        char c;
        int iCharCounter = 0;
        struct bmpData data;
        data.isPos = 1;
        data.isX = 1;
        char cFirstRow[6];
        memset(cFirstRow, '\0', sizeof(cFirstRow)/sizeof(char));
    while( ( c = fgetc( datei ) ) != EOF ){
                if( iFirstValues < 3 && ( ( 0 == iFileCounter ) || ( 0 == iFileCounter % iProcess ) ) ){
                if( ( ';' == c ) || ( ',' == c ) ){
                if( 0 == iFirstValues ){
                sscanf(cFirstRow, "%d", &iProcess);
                }
                else if( 1 == iFirstValues ){
                sscanf(cFirstRow, "%d", &iSizeY);
                }
                else if( 2 == iFirstValues ){
                sscanf(cFirstRow, "%d", &iSizeX);
                }
                memset(cFirstRow, '\0', 6);
                iFirstValues++;
            if( 3 == iFirstValues ){
            bmp = bmp_create((iPixelSize * iSizeX * 4), (iPixelSize * iSizeY * (iProcess/4)), 24);
            iFirstRow = 0;
            }
```

Quelle: [2]

Grundlagen/Wiederholung: Kommentare in C

- Zwei Arten von Kommentaren
  - Einzeiliger Kommentar (seit C99):

```c
// Dies ist ein einzeiliger Kommentar (seit C99 unterstützt)

int x = 42; //Ab hier wird die restliche Zeile als solcher betrachtet
```

Quelle: [3]

  - Blockkommentar:

```c
/* Dies ist ein Blockkommentar,
   der sich über mehrere Zeilen
   erstrecken kann. */

int x = 42 + /*Er kann auch innerhalb einer Anweisung genutzt werden */ 13 - 21;
```

Quelle: [4]

- Kommentierung kann vielseitig zur Unterstützung genutzt werden:
  - Kommentare als Zusammenfassung und Gliederung
  - Kommentare im Zusammenhang mit Funktionen
  - Kommentare bei Schleifen
  - Kommentare bei Bedingungen
  - TODO-Kommentare
  - Auskommentieren

Kommentare als Zusammenfassung und Gliederung

- „Inhaltsangabe“ mit diversen Informationen
  - Dateiname
  - Autor(en)
  - Beschreibung
  - Copyright, Lizenzen
- Gliederung durch Schlagworte, eingebettet in sich wiederholende Zeichen:

```
210  /* ************************************************************************ */
211  /* mpiInitMatrices: Init Matrix for each Process                            */
212  /* ************************************************************************ */
```

Quelle: [5]

Kommentare als Zusammenfassung und Gliederung

```
1  /* GRegex -- regular expression API wrapper around PCRE.
2   *
3   * Copyright (C) 1999, 2000 Scott Wimer
4   * Copyright (C) 2004, Matthias Clasen <mclasen@redhat.com>
5   * Copyright (C) 2005 - 2007, Marco Barisione <marco@barisione.org>
6   *
7   * This library is free software; you can redistribute it and/or
8   * modify it under the terms of the GNU Lesser General Public
9   * License as published by the Free Software Foundation; either
10  * version 2.1 of the License, or (at your option) any later version.
11  *
12  * This library is distributed in the hope that it will be useful,
13  * but WITHOUT ANY WARRANTY; without even the implied warranty of
14  * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
15  * Lesser General Public License for more details.
16  *
17  * You should have received a copy of the GNU Lesser General Public
18  * License along with this library; if not, write to the Free Software
19  * Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
20  */
```

Quelle: [6]

Kommentare im Zusammenhang mit Funktionen

- Erläuterung der Funktionalität, ggf. Hinweise zur Nutzung
- Erläuterung der Parameter
- Erläuterung/Anmerkung zum Rückgabewert

```
499 /**
500  * g_match_info_get_string:
501  * @match_info: a #GMatchInfo
502  *
503  * Returns the string searched with @match_info. This is the
504  * string passed to g_regex_match() or g_regex_replace() so
505  * you may not free it before calling this function.
506  *
507  * Returns: the string searched with @match_info
508  *
509  * Since: 2.14
510  */
511 const gchar *
512 g_match_info_get_string (const GMatchInfo *match_info)
```

Quelle: [7]

Kommentare bei Schleifen

- Erläuterung der Schleife
  - Wie viele Durchläufe?
  - Was geschieht überhaupt?
- Bei langen Schleifen aber auch als „Markierung“ am Ende
  - Strukturiert gerade bei Schachtelung den Code
  - Gibt dem Programmierer einen Überblick über größere Passagen

Kommentare bei Schleifen

```
114                 while( ( c = fgetc( datei ) ) != EOF ){
115                     if( iFirstValues < 3 && ( ( 0 == iFileCounter ) || ( 0 == iFileCounter % iProcess ) ) ){
116                         //Nur jede iProzess-te Datei darf ein neues bmp angefangen werden
117                         //& die Grund-Variablen verändert werden
118                         //in der ersten Zeile müssen dann spezielle Daten ausgelesen werden
119                         if( ( ';' == c ) || ( ',' == c ) ){
120                             //Variable kann "gefüllt" werden
121                             if( 0 == iFirstValues ){
122                                 sscanf(cFirstRow, "%d", &iProcess);
123                                 if( DEBUG ) printf("VARIABLE iProcess = %d\n", iProcess);
124                             }
125                             else if( 1 == iFirstValues ){
126                                 sscanf(cFirstRow, "%d", &iSizeY);
127                                 if( DEBUG ) printf("VARIABLE iSizeY = %d\n", iSizeY);
128                             }
129                             else if( 2 == iFirstValues ){
130                                 sscanf(cFirstRow, "%d", &iSizeX);
131                                 if( DEBUG ) printf("VARIABLE iSizeX = %d\n", iSizeX);
132                             }
133                             memset(cFirstRow, '\0', 6);
134                             iFirstValues++;
```

(etwas später…)

```
225                         }
226                         iCharCounter++;
227                         if( DEBUG_SHOW_CHARS ) printf("%d. Zeichen: %c\n", iCharCounter, c);
228                         if( DEBUG_SHOW_BMPDATA ){
229                             printf("bmpdata: X: %s | Y: %s | value: %s ", data.posX, data.posY, data.val);
230                             if( data.isPos ) printf("| isPos");
231                             if( data.isX ) printf("| isX");
232                             printf("\n");
233                         }
234                     } //while-Schleife Datei lesen
```

Quelle: [8]

Kommentare bei Bedingungen

- Ähnliche Funktionalität wie bei den Schleifen

```
190                         }
191                     } //if(zusätzliche Zeilen nach iSizeY)
192                     else{
193                         //ansonsten normale Fallunterscheidung für iSizeY Zeilen
194                         if( ':' == c ){
195                             //Wert folgt
196                             data.isPos = 0;
197                         }
198                         else if( ',' == c  || ';' == c){
199                             //neuer Datensatz bzw...
200                             if( 0 == iNewRow){
201                                 data.isPos = 1;
202                                 int iMomentaryProcess = iFileCounter % iProcess;
203                                 mySetPixel(bmp, atoi(data.val), atoi(data.posX), iRowCounter,
204                                 memset(data.posX, '\0', sizeof(data.posX)/sizeof(char));
205                                 memset(data.val, '\0', sizeof(data.val)/sizeof(char));
206                             }
207                             else iNewRow = 0;
208                             if( ';' == c) iRowCounter++;
209                             //... neue Zeile
210                         }
```

Quelle: [9]

TODO-Kommentare und Auskommentieren

- TODO-Kommentar als „provisorischer Lückenfüller“
  - Kurze Erläuterung, was noch zu tun ist
  - Teilweise Unterstützung durch die IDE (z.B. Eclipse)

- Auskommentieren
  - Oftmals die einzigen „Kommentare“, die genutzt werden ;-)
  - Codepassagen in einen Blockkommentar setzen
  - Dieser wird nicht mehr als Teil der Software interpretiert

Was ist GLib?

- GLib ist eine Bibliothek für C
- Sie enthält verschiedene, sonst nur schwer zu realisierende Funktionen
- Beinhaltet komplexe Funktionen, unter anderem in den Bereichen:
  - Makros
  - Basistypen
  - Typumwandlung
  - Timer
  - Diverse Datenstrukturen (Listen, Bäume etc.)
  - Threads
  - Diverse String-Verarbeitungsmöglichkeiten

GLib als Beispiel

```
1265 /**
1266  * g_source_set_priority:
1267  * @source: a #GSource
1268  * @priority: the new priority.
1269  *
1270  * Sets the priority of a source. While the main loop is being
1271  * run, a source will be dispatched if it is ready to be dispatched and no sources
1272  * at a higher (numerically smaller) priority are ready to be dispatched.
1273  **/
1274 void
1275 g_source_set_priority (GSource  *source,
1276                        gint      priority)
1277 {
1278   GSList *tmp_list;
1279   GMainContext *context;
1280
1281   g_return_if_fail (source != NULL);
1282
1283   context = source->context;
1284
1285   if (context)
1286     LOCK_CONTEXT (context);
1287
1288   source->priority = priority;
1289
1290   if (context)
1291     {
1292       /* Remove the source from the context's source and then
1293        * add it back so it is sorted in the correct plcae
1294        */
1295       g_source_list_remove (source, source->context);
1296       g_source_list_add (source, source->context);
1297
1298       if (!SOURCE_BLOCKED (source))
1299         {
1300           tmp_list = source->poll_fds;
1301           while (tmp_list)
1302             {
1303               g_main_context_remove_poll_unlocked (context, tmp_list->data);
1304               g_main_context_add_poll_unlocked (context, priority, tmp_list->data);
1305
1306               tmp_list = tmp_list->next;
1307             }
1308         }
1309
1310       UNLOCK_CONTEXT (source->context);
1311     }
1312 }
```

Quelle: [10]

- Funktionsbeschreibung
- Erläuterung der if-Anweisung
- Allgemein gute Codestruktur

```
2605  #ifdef G_THREADS_ENABLED
2606    if (!context->poll_waiting)
2607      {
2608  #ifndef G_OS_WIN32
2609        gchar a;
2610        read (context->wake_up_pipe[0], &a, 1);
2611  #endif
2612      }
2613    else
2614      context->poll_waiting = FALSE;
2615
2616    /* If the set of poll file descriptors changed, bail out
2617     * and let the main loop rerun
2618     */
2619    if (context->poll_changed)
2620      {
2621        UNLOCK_CONTEXT (context);
2622        return FALSE;
2623      }
2624  #endif /* G_THREADS_ENABLED */
```

Quelle: [11]

- Abschließende Markierung bei Bedingungen
- Hier: Kleine Schachtelung
- Gerne bei Präprozessor-Anweisung genutzt

```
3137                         g_print ("]");
3138                       }
3139                     i++;
3140                   }
3141                 pollrec = pollrec->next;
3142               }
3143             g_print ("\n");
3144
3145             UNLOCK_CONTEXT (context);
3146           }
3147   #endif
3148       } /* if (n_fds || timeout != 0) */
```

Quelle: [12]

- Als Abgrenzung für Bedingungen
- Hier: Wiederholung der Bedingung

```
2711         /* attribute not collected:  could be caused by two things.
2712          *
2713          * 1) it doesn't exist in our list of attributes
2714          * 2) it existed but was matched by a duplicate attribute earlier
2715          *
2716          * find out.
2717          */
```

Quelle: [13]

- Erläuterung der momentanen „Situation“
- Erläuterung, wie jetzt fortgefahren wird

Quelltextdokumentationssysteme – eine Übersicht

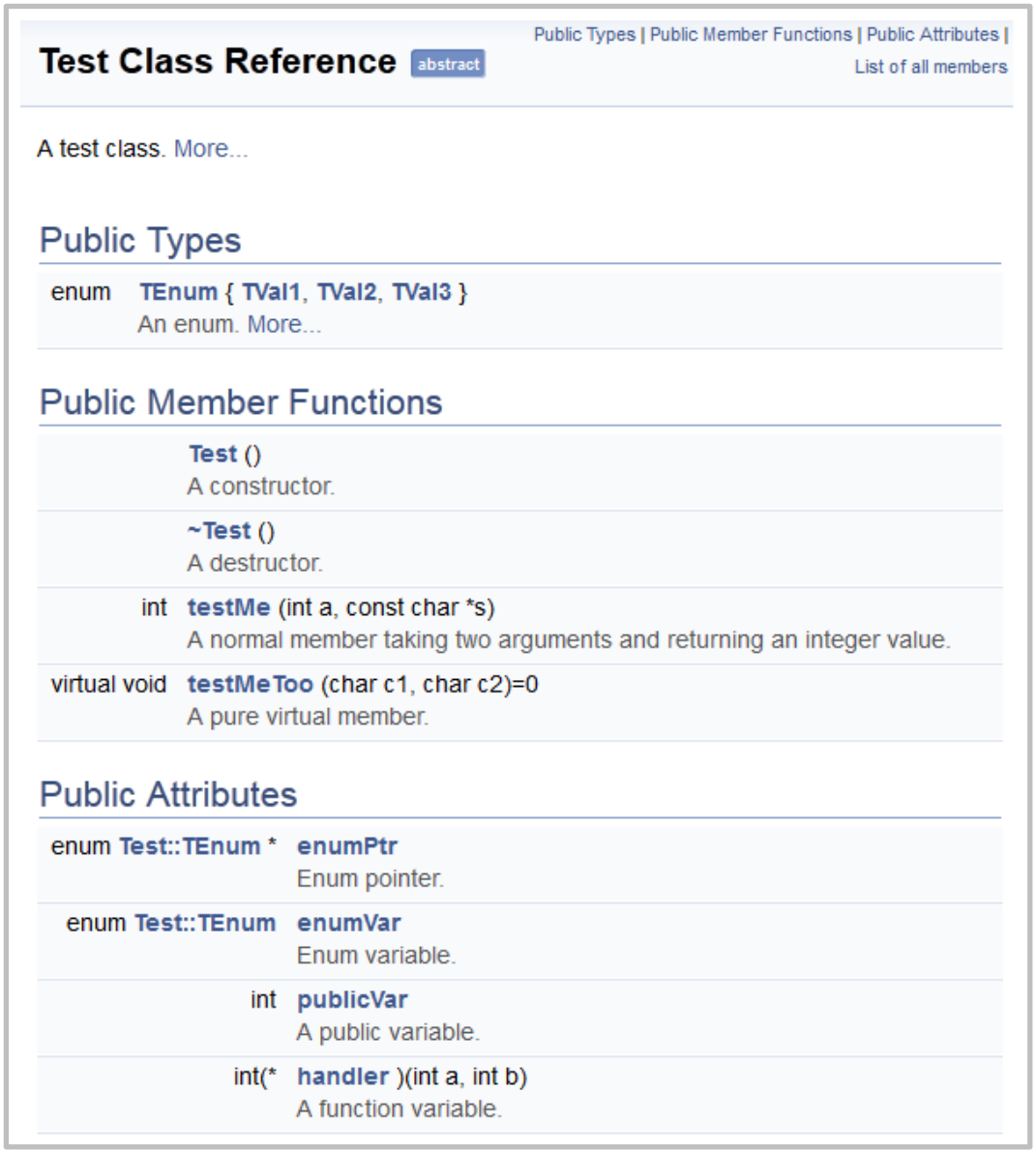
- Software zur automatischen Erzeugung von Dokumentationen
- Nutzt Quelltext, UML-Diagramme und Grafiken zur Erzeugung
- Verschiedene Ausgabeformate: html, LaTeX, XML, PDF und andere
- Beispiele:
  - Javadoc (Java)
  - Doxygen (C, C++, Objective-C, Java, Python, Fortran)
  - GTK-Doc (C)
  - Natural Docs (Perl, C#, ActionScript)

Dokumentationssysteme

```
1  //!  A test class.
2  /*!
3    A more elaborate class description.
4  */
5
6  class Test
7  {
8    public:
9
10     //! An enum.
11     /*! More detailed enum description. */
12     enum TEnum {
13                  TVal1, /*!< Enum value TVal1. */
14                  TVal2, /*!< Enum value TVal2. */
15                  TVal3  /*!< Enum value TVal3. */
16                }
17          //! Enum pointer.
18          /*! Details. */
19          *enumPtr,
20          //! Enum variable.
21          /*! Details. */
22          enumVar;
23
24     //! A constructor.
25     /*!
26       A more elaborate description of the constructor.
27     */
28     Test();
29
30     //! A destructor.
31     /*!
32       A more elaborate description of the destructor.
33     */
34    ~Test();
35
36     //! A normal member taking two arguments and returning an integer value.
37     /*!
38       \param a an integer argument.
39       \param s a constant character pointer.
40       \return The test results
41       \sa Test(), ~Test(), testMeToo() and publicVar()
42     */
43     int testMe(int a,const char *s);
```

Quelle: [14]

- Beispiel: Doxygen (C++)
- Einfache Testklasse mit Doxygen-Notation der Kommentare
- Eigentlicher Quelltext deutlich schwerer zu lesen



Public Types | Public Member Functions | Public Attributes | List of all members

Test Class Reference abstract

A test class. More...

Public Types

enum **TEnum** { **TVal1**, **TVal2**, **TVal3** }
An enum. More...

Public Member Functions

Test ()
A constructor.

~Test ()
A destructor.

int **testMe** (int a, const char *s)
A normal member taking two arguments and returning an integer value.

virtual void **testMeToo** (char c1, char c2)=0
A pure virtual member.

Public Attributes

enum **Test::TEnum** * **enumPtr**
Enum pointer.

enum **Test::TEnum** **enumVar**
Enum variable.

int **publicVar**
A public variable.

int(* **handler**)(int a, int b)
A function variable.

Quelle: [15]

- Ausgabe im html-Format
- Übersichtliche Auflistung
  - Übersichten
  - Detailierte Angaben
  - Index

Zusammenfassung

- Kommentierung wird oftmals unterschätzt und vernachlässigt
- Es *kann* die Arbeit an Software deutlich vereinfachen
- Kommentare sollten sinnvoll gewählt werden
- Man kann Kommentierung sehr flexibel nutzen
- Die Möglichkeiten lassen sich auch auf andere Sprachen übertragen
- Kommentierung ist in Team-Projekten unverzichtbar

Quellen

- *http://de.wikipedia.org/wiki/Kommentar_%28Programmierung%29*
- *http://en.wikipedia.org/wiki/Comment_%28computer_programming%29*
- *http://queue.acm.org/detail.cfm?id=1053354*
- *http://de.wikipedia.org/wiki/Software-Dokumentationswerkzeug*
- *Eigene Erfahrungen ;-)*

- Codebeispiele [1], [3], [4]: Eigenes Beispiel
- Codebeispiele [2], [5], [8], [9]: Eigene Arbeit
- Codebeispiele [6] + [7]: GLib Version 2.26.1 (Windows), *\glib-2.26.1\glib\gregex.c*
- Codebeispiele [10], [11], [12], [13]: GLib Version 2.26.1 (Windows), *\glib-2.26.1\glib\gmain.c*
- Codebeispiele [14] + [15]: Doxygen Beispiel in C++, Qt Style
- Alle Codebeispiele wurden mit *QtCreator 2.2.0* noch einmal überarbeitet