HOCHSCHULE
HAMM-LIPPSTADT

Programmierrichtlinie

für die Erstellung von Software in C

von:	Prof. Dr. Ulrich Schneider
Stand:	10. Oktober 2022
Version:	1.1

Inhaltsverzeichnis

1 Einleitung

Dieses Dokument ist ein Leitfaden für die Namenskonventionen beim Programmieren in C und C++ von Prof. Schneider.

2 Modul- und Funktionsköpfe

Für die Verständlichkeit der implementierten Software sind Kommentare und Beschreibungen von Modulen und Funktionen unerlässlich. Um einen schnellen Überblick über die vorliegende Implementierung zu erhalten, werden Module durch einen Modulkopf eingeleitet. Dieser befindet sich stets zu Beginn des Moduls. Selbiges gilt für die Beschreibung von Funktionen. Vorlagen für Modul- und Funktionsköpfe werden im Folgenden für verschiedene Programmiersprachen vorgestellt. Die Beispiele können aus diesem Dokument kopiert werden. Wichtig ist jedoch die gewissenhafte Änderung der Kopf-Einträge.

Ein Modulkopf für eine Implementierung in C oder C++ wird in Quelltext 1 gezeigt. Dieser wird eingeleitet vom Modulnamen, gefolgt vom Erstellungsdatum. Dieses Datum darf nach der erstmaligen Implementierung des Moduls nicht mehr verändert werden. In der Modulbeschreibung wird der Zweck des Moduls im Stil eines Titels genannt. Anschließend wird die verwendete Entwicklungsumgebung und der Name des Autors aufgeführt. Außerdem können Bemerkungen zum Modul hinzugefügt werden. Abschließend wird das Datum der letzten Änderung genannt.

Quelltext 1: Strukturierung eines Modulkopfs in C oder C++.

```
/************************************************************\
*
* Modul           : ModulName.c
*
* Datum           : 04. Oktober 2013
*
* Beschreibung    : Zweck dieses Moduls
*
* Implementierung : Visual Studio 2012 Professional
*
* Autor           : Mustermann, Max
*
* Bemerkung       : Demo für den ersten Meilenstein
*
* Letzte Änderung : 04. Mai 2018
*
\************************************************************/
```

Funktionen werden innerhalb eines Moduls implementiert. Der Funktionskopf ist ähnlich wie ein Modulkopf aufgebaut. Zunächst wird der Funktionsname genannt. Bei der Wahl des Funktionsnamens müssen die Bedingungen der Namenskonventionen beachtet werden. Die folgenden Einträge werden wie vorangegangen beschrieben ausgefüllt.

Der Kopf einer Funktion unterscheidet sich im Wesentlichen von dem eines Moduls durch die Beschreibung der übergebenen und der zurückgegebenen Funktionsparameter. Die Beschreibung der Übergabeparameter erfolgt in tabellarischer Form wie im Quelltext 2 gezeigt. Es wird der Datentyp, Name und eine Beschreibung des Parameters genannt. Ebenso werden der Datentyp und die Beschreibung des Rückgabeparameters der Funktion genannt.

Quelltext 2: Strukturierung eines Funktionskopfs in C und C++.

```
/************************************************************\
*
* Funktion          : MD_FunktionsName
*
* Datum             : 04. Oktober 2013
*
* Beschreibung      : Zweck dieser Funktion
*
* Implementierung   : Visual Studio 2012 Professional
*
* Autor             : Mustermann, Max
*
* Bemerkung         : Code-Review noch ausstehend
*
* Letzte Änderung   : 04. Mai 2018
*
* Übergebeparameter :
* Typ      Name       Beschreibung
* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* int      n          Anzahl Elemente des Arrays
* double[] a          Array mit double-Werten
*
* Rückgabeparameter :
* Typ      Beschreibung
* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
* int      Rückgabe eines Fehlercodes
*
\************************************************************/
```

3 Namenkonventionen von Variablen

Aufbau von Variablen:

Beschreibung	**Modulname**	**Beschreibung**	**_**	**Typ**
Inhalt	Kürzel des Modulnamens. 2 – 5 Buchstaben.	Aussagekräftige Beschreibung des Inhaltes	Unterstrich	u8, s8, u16, s16, u32, s32, st (struct), bit (bit)
Beispiel	**MD_**ErrCount_u8	MD_**ErrCount**_u8	MD_ErrCount**_**u8	MD_ErrCount_**u8**
Bemerkung	projektspezifisch einheitlich, im Datenlexikon definiert. Gilt nicht für lokale Funktionen und lokale Variablen.	Verwendung einheitlicher Bezeichner	Entfällt bei einer Variable die von einem typedef abgeleitet ist.	Entfällt bei einer Variable die von einem typedef abgeleitet ist. Wenn bei der Definition einer struct, enum oder union ein tag deklariert wird, bekommt der Tagname das Suffix tag angehäng t.

Hinweis: In kleinen Programmierprojekten kann das Modulkürzel entfallen.

3.1 Namen von Funktionen

Funktionsnamen werden durch eine Aneinanderreihung von Wörtern aufgebaut, wobei in jedem dieser Wörter der erste Buchstabe groß geschrieben wird und der Rest klein. Werden Funktionen außerhalb eines Moduls verwendet, wird das Modulkürzel vorangestellt, handelt es sich um lokale Funktionen wird der erste Buchstabe klein geschrieben. Funktionen bekommen keinen Typen-Suffix.

Beispiele:

```
void MD_MisalignmentDetection(void);
static BIT getRawData (void);
```

3.2 Namen von Konstanten

Unter Konstanten werden an dieser Stelle die Definitionen für den Präprozessor (per `#define`) verstanden. Über die Forderung nach einer aussagekräftigen Wahl des Namens hinaus gelten folgende Regeln:

1. Symbolische Konstanten werden durchgehend mit Großbuchstaben beschrieben, wobei längere Bezeichnungen zur besseren Lesbarkeit durch Unterstriche gegliedert werden.

2. Analog zu der Namenskonvention für Variablen wird auch an die Konstante der Typ angehängt. Der angegebene Typ wird dabei durch den Wertebereich vorgegeben, den diese Konstante annehmen kann.

3. Ein Modulpräfix sollte bei globalen Konstanten Aufschluss über den Definitionsort geben.

Beispiel:

```
#define MD_MAX_RAWCHANNEL_TRANSITIONS_u8 ((u8)20) /* globale Konstante */
static const u16 ENOUGH_DATA_LIMIT_u16 = 10;  /* modullokale Konstante */
```

3.3 Namen von Datentypen

Alle vom Benutzer definierten Typen, also diejenigen die mit `typedef` definiert sind, müssen mit der Endung `_t` versehen werden. Der Name von Datentypen wird wie bei Funktionsnamen aus Wörtern gebildet, deren erster Buchstabe groß geschrieben wird.

Beispiel:

```
typedef struct
{
UncertaintyStruct_t Uncertainty_st; // uncertainty of result
s16 Value_s16; // resulting value
}
ResultValueStruct_t;
```

3.4 Namen von Variablen

Diese Notation legt die Benennung der Variablen bezüglich ihres Typs fest. Der Name wird wie bei Funktionsnamen aus Wörtern gebildet, deren erster Buchstabe groß geschrieben wird. Lokale Variablen beginnen mit einem Kleinbuchstaben und globale mit dem Modulpräfix. Als Suffix ist einer der folgenden Typenbezeichner getrennt mit einem Unterstrich zu verwenden:

u8	unsigned Char (8 Bit Integer ohne Vorzeichen)	[0,255]
s8	signed Char (8 Bit Integer mit Vorzeichen)	[−128,+127]
u16	Word (16 Bit Integer ohne VZ)	[0, 65 535]
s16	Word (16 Bit Integer mit VZ)	[−32 768, +32 767]
u32	Double-Word (16 Bit Integer ohne VZ)	[0, 4 294 967 295]
s32	Double-Word (16 Bit Integer mit VZ)	[−2 147 483 648, +2 147 483 647]

Empfehlung (nicht bindend):

bit	Bit (FALSE, TRUE)	[0,1]
p	Zeiger (pointer) auf nachfolgenden Typ	
a	Feld (array) vom nachfolgenden Typ	
st	Struktur	-

Beispiele:

```
u16 MD_StdAlignmentAngle_u16;              /* globale Variable             */
static TrackStruct_t TrackList_ast[25];    /* modulglobales Feld ein Struktur */
static BIT StartupWithDTC1678Active_bit;   /* modulglobale Variable    */
TrackStruct_t* TrackList_pst;              /* Zeiger auf eine Struktur     */
BIT channelTransValid_bit;                 /* lokale Variable vom Typ BIT  */
```

4 Sonderregeln

4.1 Schleifeninkrement

Array Indices dürfen in einfachen For-Schleifen zur Übersichtlichkeit mit einfachen Zählervariablen belegt werden. In komplexen oder verschachtelten Schleifen sollten jedoch sprechenden Namen verwendet werden. Beispiel:

```
for (i=1;i<3;i++)
{
  Memory[i]=0;
}
```

besser

```
for (index_u8=1; index_u8<3; index_u8++)
{
  Memory[index_u8]=0;
}
```

Weitere Beispiele fur Indexbezeichner sind:

```
cnt_u8, trackCnt_u8, channelNr_u8
```

5 Wortwahl

Die Wortwahl kann durch einige Basisregeln erleichtert werden

- Variablennamen müssen aus Wörtern der deutschen Sprache gebildet werden.
- Die Variablen müssen einfach lesbar und verständlich sein. Die Syntax der englischen Worte HorizontalAlignment ist beispielsweise einfacher lesbar als AlignmentHorizontal.
- Lesbarkeit geht vor Kürze . CalculateHorizontalAlignment ist beispielsweise verständlicher als CalPsi (Ψ ist hier die schwer verständliche Abkürzung für den horizontalen Ausrichtungswinkel).
- Unterstriche dienen nur zur Trennung bei Konstantennamen oder zur Abgrenzung des Datentypen Suffix.
- Bindestriche und andere nicht alphanumerische Zeichen (Buchstaben und Ziffern) sind zu vermeiden.
- Die zuvor beschriebene Variante der ungarischen Notation ist zu verwenden.
- Bezeichnungen, die in Konflikt mit gängigen Bezeichnern von Programmiersprachen stehen, sind zu vermeiden, z. B. `sprintf`, `if`, `for`, `while` `etc.`