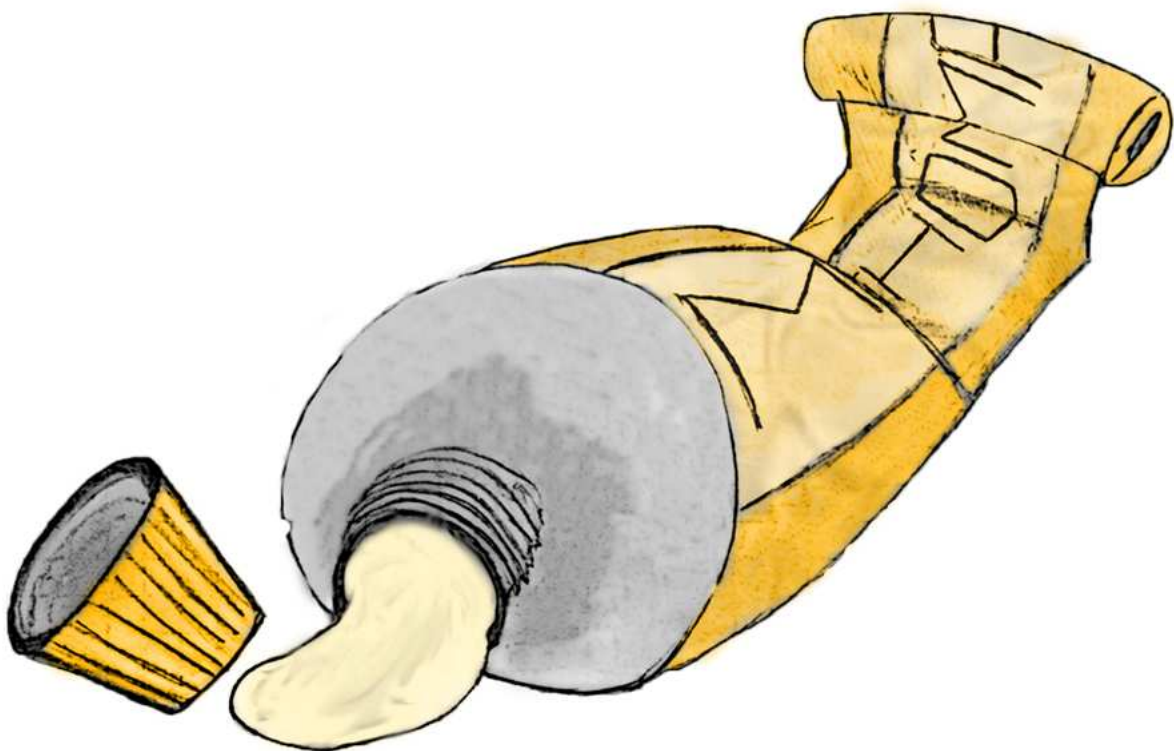


# Programmier- richtlinien

## „MathoMadaja“



# MATHOMADAJA

1	<b>Einführung</b> .....	<b>3</b>
2	<b>Allgemeines</b> .....	<b>3</b>
3	<b>Programmaufbau</b> .....	<b>4</b>
4	<b>Variablendefinitionen und Deklarationen</b> .....	<b>6</b>
5	<b>Quellen</b> .....	<b>7</b>

## 1 Einführung

Der Zweck eines definierten Programmierstils ist die Erleichterung der Arbeit aller an einem Programmierprojekt beteiligten Teammitglieder. Das bezieht sich insbesondere auf:

- Lesbarkeit,
- Verständlichkeit,
- Wartbarkeit,
- Eliminierung vermeidbarer Fehlerquellen.

Im Sinne der Verständlichkeit und Wartbarkeit kann eine Richtlinie die Verwendung von programm Sprachlich erlaubten, aber unsauberen Programmkonstrukten einschränken oder ganz verbieten. Die Einhaltung von vorgängig definierten Nomenklaturen für Variablen, Prozeduren und Klassennamen kann Lesbarkeit und Wartbarkeit eines Programmcodes wesentlich verbessern.

Während der Wartung ist die Einhaltung eines definierten Programmierstils noch wichtiger als während der Entwicklung. Als Richtwert gilt, dass 80 % der Lebenszeit eines Softwareprodukts auf die Wartung entfallen. Nur selten wird ein Produkt vom ursprünglichen Programmierer gewartet. Umso wichtiger ist es, dass bereits vom ersten Augenblick an ein guter Programmierstil verwendet wurde.

Ein Programmierstil sollte nicht unbedingt wie eine Doktrin ausgelegt werden. Verstöße gegen einen Programmierstil sollten erlaubt sein, wenn sie gut begründet sind. Dies kann in Einzelfällen beispielsweise (beim Programmierstil im engeren Sinne) durch optimierte Platzausnutzung den Überblick verbessern, durch Betonung bestimmter Einzelheiten der Verständlichkeit dienen oder als Ad-hoc-Sonderregel für besondere Codeteile die Ziele des Programmierstils mit anderen Mitteln verfolgen.

## 2 Allgemeines

- Das Programm soll im C-Standard programmiert werden. Auf Hilfsmittel, wie z.B. spezielle C++-Methoden und Klassen muss daher verzichtet werden. Eine Ausnahme bilden hier die Klassen, Methoden und Eigenschaften der grafischen Entwicklungsumgebung Qt. Werden Objekte der Qt-Klassen benötigt, so werden diese gesondert in Header- und Sourcefiles angelegt und verwendet. Das Hauptprogramm benötigt zur Darstellung der grafischen Oberfläche die Implementierung der Qt-Bibliotheken.
- Speziallösungen sind zu vermeiden (möglichst Standardfunktionen verwenden).
- Defensiv programmieren (auch unmögliche Fälle abfangen). Eine switch-Anweisung erhält z. B. auch dann einen default-Zweig, wenn die augenblickliche Programmversion alle möglichen Fälle abdeckt. Auf diese Weise werden bei späteren Programmweiterungen Fehler u. U. schneller

gefunden, wenn die Erweiterung auch in der switch-Anweisung berücksichtigt werden muss (aber nicht berücksichtigt worden ist).

- Bei fehlerhaften Eingaben wird ein Hilfetext (MessageBox) ausgegeben, der den Fehler und die gewünschte Eingabe erläutert, bevor die Eingabe erneut angefordert wird.
- Typen müssen explizit umgewandelt werden, falls Typunterschiede vorliegen  
Beispiel: `int *iPointer = (int *) malloc (sizeof(int));`

### 3 Programmaufbau

- Jede Datei beginnt mit einem Kopf, der folgenden Aufbau hat:

```
/* Beschreibung des Dateiinhalts und wichtige
Bemerkungen.
*
* Datei: <Dateiname> Autor: <Name>
* Datum: <...>
*
* Historie: <WER hat WANN WAS geaendert / ergaenzt>
*/
```
- Nach dem Dateikopf wird folgende Programmstruktur verwendet:
  1. Anweisungen an den Präprozessor:
    - a. Angabe der benötigten Header-Dateien
    - b. Symbolische Konstanten und Makros definieren
  2. Globale Variablen deklarieren (wenn dringend benötigt)
  3. neue (eigendefinierte) Typen (structs) definieren
  4. Funktionsprototypen deklarieren
  5. Funktionen definieren (die erste Funktion ist die Funktion main!)
- Jede Funktion (Prozedur) beginnt ebenfalls mit einem Kopf, der mindestens eine Funktionsbeschreibung, sowie eine Beschreibung der Ein- und Ausgabeparameter, des Rückgabewertes und der Fehlerausgänge enthält. Eventuell können auch Angaben zur Laufzeit- und Speicherplatzkomplexität hinzugefügt werden. Darüber hinaus sind Seiteneffekte (Änderung globaler Variablen) unbedingt anzugeben. Folgendes Kommentierschema beachten:

```
/**
* Funktionsbeschreibung: Dies ist eine Funktion die was
* macht.
*
* Eingabeparameter: ID der Eingabe, String Quelle, String
* Ziel
*
* Rückgabe: Fehlerstatus der Funktion, Erfolg oder Fehler.
*
* Besonderes: ...
*/
```

- Das Programm wird durchgehend in deutsch kommentiert. Es werden nur sinnvolle Kommentare verwendet, die nicht unmittelbar aus dem Programmtext erkennbar sind. Kommentare sollen das Programm leichter lesbar machen. Auch hier gilt, Kommentare im C-Standard Wichtig für die Dokumentation ist folgendes Kommentierschema zu beachten:

```
/**  
 * Kommentar  
 * 2. Zeile ...  
 */
```

Was nicht in die Dokumentation soll wird ohne einleitende \*\* kommentiert. Also so z.B:

```
/* Dies ist ein Kommentar, welches so nicht in der  
Dokumentation erscheint. */
```

- Jeder Block wird entsprechend der Blocktiefe sauber und lesbar eingerückt.

Beispiel:

```
if (...)
{
    Anweisung(en);
}
else
{
    if(...)
    {
        Anweisung(en);
    }
}
```

- Die geschweiften Klammern werden auch dann verwendet, wenn nur eine Anweisung vorhanden ist!  
Operatoren und Operanden werden durch Leerzeichen voneinander getrennt.  
Beispiele: `x = a + b; z = ((x < y) ? x : y);`

- Ausdrücke werden ggf. durch Klammern strukturiert (besser lesbar).

Beispiel:

```
while ((table[i][0] != c) && (i < tab_len))
```

Auch hier gilt wieder den Code selbstsprechend zu machen, also keine Abkürzungen in Bedingungen!

- Je Zeile ist nur eine Anweisung erlaubt!
- Es sollte darauf geachtet werden, dass die Länge einer Anweisung die Bildschirmbreite nicht überschreitet. Wenn möglich wird die Zeile also umgebrochen (Einrücken dann nicht vergessen). Trägt das Umbrechen einer Zeile nicht zur besseren Lesbarkeit des Codes bei, so wird die Anweisung in eine Zeile geschrieben.

## 4 Variablendefinitionen und Deklarationen

- Deklarationen erfolgen so lokal wie möglich (Modularisierung, Information Hiding).
- Namen für Variablen müssen sorgfältig gewählt und sollten, soweit nicht sprechend, als Kommentar näher erläutert werden. Zur Strukturierung der Namen wird eine Mischung aus Groß- und Kleinbuchstaben verwendet. Namen sollten natürlich nicht zu lang gewählt werden. Jeder Variablenname beginnt mit einem Indiz für den Variablentyp. Danach folgt der beschreibende Name, beginnend mit Großbuchstaben, und kann dann z.B. auch mit „\_“ variiert werden.

### Beispiel für Standard-Datentypen:

int	i
unsigned int	uint
short int	sint
unsigned short int	usint
long int	lint
unsigned long int	ulint
char	c
unsigned char	uch
signed char	c
char *	str
float	f
double	d
long double	ld

```
int iGanzzahl;
char cZeichen;
double dPointer_Array;
...
```

- Dasselbe gilt für Strukturen und Auswahlfelder.
- Dasselbe gilt für die grafischen Qt-Objekte. Deren Namen sollten ebenso mit einem Indiz für den Objekttyp beginnen.
- Namen in typedef und #define Anweisungen werden im Allgemeinen in Großbuchstaben geschrieben.
- Es werden bevorzugt deutsche Namen verwendet, außer der Name ist in einer anderen Sprache kürzer und verständlicher.  
Bsp.: *int iSizeList*, besser als *int iGroesseListe*.

## 5 Quellen

- <http://de.wikipedia.org/wiki/Programmierrichtlinie>
- <http://www.gnu.org/prep/standards/>
- <http://www-home.fh-konstanz.de/~bittel/prog/style.pdf>
- [http://www2.fh-fulda.de/~gross/c\\_richt.pdf](http://www2.fh-fulda.de/~gross/c_richt.pdf)