

Objektorientierte Programmierung

Code Style

Name: Jörg Sahm
Raum: 4.2.10
E-Mail: joerg.sahm@fh-erfurt.de
Tel.: 6700 - 352

- Definition zur Formatierung des Quellcodes, etwa hinsichtlich ...

- ... Einrückungen

// Variante A

```
if (a > b)
{
    if (a > c)
    {
        b = b * c;
    }
}
```

- ... Arithmetische Ausdrücke

- ... Klammerblöcke

// Variante B

```
if (a>b)
if (a>c)
b=b*c;
```

Vereinfachung?

- ... besseres Verständnis des Codes
- ... schnelleres Zurechtfinden in fremder Software
- ... eine Systematik, in der Fehler schneller auffallen
- ... eine Ästhetik der Software und damit höhere Motivation

```
if (X = 7)
{
    printf("X is 7\n");
}

if (X == 7);
{
    printf("X is 7\n");
}

while (X != 5);
{
    ++ X;
}

for (int Index = 0; Index < 10; ++ Index);
{
    printf("Index: %i\n", Index);
}

{
    int A, B;

    B = 0;

    int Sum = A + B;
}
```

```
if (X = 7)
{
    printf("X is 7\n");
}

if (X == 7) ;
{
    printf("X is 7\n");
}

while (X != 5) ;
{
    ++ X;
}

for (int Index = 0; Index < 10; ++ Index) ;
{
    printf("Index: %i\n", Index);
}

{
    int A, B;

    B = 0; A = 5;

    int Sum = A + B; // Erzeugt Compilerwarnung
}
```

// Variante A

```
if      (X == 1) printf("X is 1\n");  
else if (X == 2) printf("X is 2\n");  
else if (X == 3) printf("X is 3\n");  
else if (X == 3) printf("X is 4\n");
```

// Variante B

```
if (X == 1) printf("X is 1\n"); else  
    if (X == 2) {  
        printf("X is 2\n"); }  
else if (X == 3) printf("X is 3\n"); else if (X == 3) printf("X  
is 4\n");
```

// Variante A

```
if(A== 1) {printf("A is 1\n");  
do {  
if (B>5) C =B-A/2;  
for(int I=0;I < 10;++I)  
    {  
A= B * C +7;  
}}  
while (C < 7) if ((B==A)&&(A>7)) {  
printf("Value: %i %i %i\n", A, B, X);}
```

Wo sind die drei Fehler?

```
// Variante B
```

```
if (A == 1)
{
    printf("A is 1\n");

    do
    {
        if (B > 5) C = B - A / 2;

        for (int I = 0; I < 10; I ++ )
        {
            A = B * C + 7;
        }
    }
    while (C < 7)

    if ((B == A) && (A > 7))
    {
        printf("Value: %i %i %i\n", A, B, C);
    }
}
```

Wo sind die drei Fehler?


```
// Variante B
```

```
if (A == 1)
{
    printf("A is 1\n");

    do
    {
        if (B > 5) C = B - A / 2;

        for (int I = 0; I < 10; I ++ )
        {
            A = B * C + 7;
        }
    }
    while (C < 7);

    if ((B == A) && (A > 7))
    {
        printf("Wert: %i %i %i\n", A, B, C);
    }
}
```

Wo sind die drei Fehler?

```
for (i = 0; i < n; i ++)  
{  
    const UByte4& c1 = B1[i];  
  
    x = static_cast<int>(c1.x) * c.x / 255;  
    y = static_cast<int>(c1.y) * c.y / 255;  
    z = static_cast<int>(c1.z) * c.z / 255;  
    w = static_cast<int>(c1.w) * c.w / 255;  
  
    x = Clamp(x, 0, 255);  
    y = Clamp(y, 0, 255);  
    z = Clamp(z, 0, 255);  
    w = Clamp(w, 0, 255);  
  
    UByte4& c2 = B2[i];  
  
    c2.x = static_cast<U8>(x);  
    c2.y = static_cast<U8>(y);  
    c2.z = static_cast<U8>(z);  
    c2.w = static_cast<U8>(w);  
}
```

```
for (PixelFormat = 0; PixelIndex < PixelCount; PixelIndex ++)  
{  
    const UByte4& rSrcColor = pSrcBuffer[PixelIndex];  
  
    Alpha = static_cast<int>(rSrcColor.m_Alpha) * _rColor.m_Alpha / 255;  
    Red    = static_cast<int>(rSrcColor.m_Red   ) * _rColor.m_Red   / 255;  
    Green  = static_cast<int>(rSrcColor.m_Green) * _rColor.m_Green / 255;  
    Blue   = static_cast<int>(rSrcColor.m_Blue  ) * _rColor.m_Blue  / 255;  
  
    Alpha = Clamp(Alpha, 0, 255);  
    Red    = Clamp(Red, 0, 255);  
    Green  = Clamp(Green, 0, 255);  
    Blue   = Clamp(Blue, 0, 255);  
  
    UByte4& rDstColor = pDstBuffer[PixelIndex];  
  
    rDstColor.m_Alpha = static_cast<U8>(Alpha);  
    rDstColor.m_Red   = static_cast<U8>(Red);  
    rDstColor.m_Green = static_cast<U8>(Green);  
    rDstColor.m_Blue  = static_cast<U8>(Blue);  
}
```

- Aussagekräftige Bezeichner verwenden
 - Keine ungeeigneten Kürzel
- Erfahrene Programmierer schreiben Bezeichner oft nur einmal
 - Copy & Paste
 - Tomato Visual Assist
 - Intellisense unter C++ nicht wirkungsvoll

- Kurz und bündig
- Keine zu extremen Vorgaben ???
- Für jede Vorgabe ein richtiges und ein falsches Beispiel
- Maximale Aussagekraft mit **minimalen** Aufwand
 - Schlechtes Beispiel **Ungarische Notation**

- Code Style ist das A und O
 - Das Einhalten eines Code Styles kostet Energie
 - Wer einen Code Style pflegt, gibt sich Mühe bei seiner Software
 - Erste wichtige Erkenntnis bei fremder Software
 - Die investierte Energie zahlt sich aus
 - Mehr Übersicht
 - Mehr Spaß am Code (wichtig vor allem bei mehrjährigen Projekten)
 - Schnelles Zurechtfinden im – eigenen – Code auch nach Pausen
 - Leichtere Fehlersuche
 - Weniger Fehlerquellen
 - Zeitgewinn

- Es gibt sicherlich gute und schlechte Code Styles
 - Wichtig ist erst einmal, **überhaupt** einen Code Style zu haben
 - Nichts ist schlimmer, als Software ohne jeden Code Style
 - Ein guter Programmierer kommt mit jedem guten Code Style zurecht
- Kein Fan der ungarischen Notation
 - Viele Präfixe und Kombinationen von Präfixen
 - Variablennamen werden kryptischer
 - Variablennamen werden länger

- Es gibt immer Primadonnen, die sich aus Prinzip selbst an **gute** Firmenvorgaben nicht halten, und ihr eigenes Ding machen (nicht nur beim Code Style)
 - Argumentation Anhören
 - Eigene Fehler einsehen
 - Begründen der Vorgaben
 - Im schlimmsten Fall rausschmeißen!!!



- Gegen Ende des Projekts kommen oftmals Panikeinstellungen neuer Programmierer
 - Zu viele Features fehlen noch
- Einige halten sich nicht an den Code Style
 - Leads fehlt die Zeit, auch noch darauf zu achten
 - Hinweis auf Code Style erfolgt erst gar nicht
 - Schwierig, darauf noch zu reagieren
 - Projekt steht auf dem Spiel

- Arbeiten viele Leute an einer Software, so wird das persönliche „Kunstwerk“ zerstört
 - Sie bearbeiten und pflegen eine Klasse über mehrere Jahre
 - Anderer Mitarbeiter baut eigenen Code ein
 - Code ist im Rahmen der Vorgaben, aber definitiv anders als der eigene
- Jahrelange Pflege ist innerhalb weniger Minuten hin
- Hm, keine zu extremen Vorgaben beim Code Style?
 - Firmen geben durchaus Mitarbeitern ganze Ordner mit Vorschriften