

# Vertiefung Software-Entwicklung in C++

Prof. Dr. rer. nat. Peter Gerwinski

8. Oktober 2018

# Vertiefung Software-Entwicklung in C++

Prof. Dr. rer. nat. Peter Gerwinski

*rerum naturalium* = der natürlichen Dinge (lat.)

8. Oktober 2018

# Vertiefung Software-Entwicklung in C++

Prof. Dr. rer. nat. Peter Gerwinski

*rerum naturalium* = der natürlichen Dinge (lat.)

8. Oktober 2018

# Zu dieser Lehrveranstaltung



- **Lehrmaterialien:**  
<https://gitlab.cvh-server.de/pgerwinski/cpp.git>
- **Statt Klausur: Projektaufgabe**
  - Hausarbeit und Kolloquium
  - ein neues, nichttriviales Programm in einer C++-ähnlichen Sprache selbst entwickeln
  - ein vorhandenes, nichttriviales Programm in einer C++-ähnlichen Sprache mit- und/oder weiterentwickeln
  - Sonstiges, was zum Thema der Lehrveranstaltung paßt und sich auf Master-Niveau bewegt
- **Plan:**  
die Theorie möglichst zügig abarbeiten,  
möglichst frühzeitig mit dem praktischen Arbeiten beginnen
- **Wir sind flexibel. ;–)**



# Vertiefung Software-Entwicklung in C++

<https://gitlab.cvh-server.de/pgerwinski/cpp.git>

## 1 Einführung

1.1 Was ist C?

1.2 Was ist C++?

## 2 Wiederholung: Programmieren in C

## 3 Einführung in C++

## 4 Standard-Bibliotheken (STL)

## 5 C++11

## 6 Plug-In-Architekturen

## 7 Die Boost-Bibliothek



Änderungen  
vorbehalten

# 1 Einführung

## 1.1 Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen



Hardware und/oder Betriebssystem

- Hardware direkt ansprechen und effizient einsetzen
- ... bis hin zu komplexen Software-Projekten

→ Man kann Computer vollständig beherrschen.

# 1 Einführung

## 1.1 Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- Hardware direkt ansprechen und effizient einsetzen
- ... bis hin zu komplexen Software-Projekten
- leistungsfähig, aber gefährlich

„High-Level-Assembler“

- kein „Fallschirm“
- kompakte Schreibweise

*C makes it easy to shoot yourself in the foot.*

Bjarne Stroustrup, ca. 1986

[http://www.stroustrup.com/bs\\_faq.html#really-say-that](http://www.stroustrup.com/bs_faq.html#really-say-that)

Unix-Hintergrund

- Baukastenprinzip
- konsequente Regeln
- kein „Fallschirm“

# 1 Einführung

## 1.2 Was ist C++?

Etabliertes Profi-Werkzeug

- kompatibel zu C

C++ unterstützt

- *objektorientierte Programmierung*
- *Datenabstraktion*
- *generische Programmierung*

*C++ is a better C.*

Bjarne Stroustrup, Autor von C++  
<http://www.stroustrup.com/C++.html>

*C makes it easy to shoot yourself in the foot;  
C++ makes it harder, but when you do  
it blows your whole leg off.*

Bjarne Stroustrup, Autor von C++, ca. 1986  
[http://www.stroustrup.com/bs\\_faq.html](http://www.stroustrup.com/bs_faq.html)  
[#really-say-that](#)



# Vertiefung Software-Entwicklung in C++

<https://gitlab.cvh-server.de/pgerwinski/cpp.git>

## 1 Einführung

1.1 Was ist C?

1.2 Was ist C++?

## 2 Wiederholung: Programmieren in C

2.1 Hello, world!

2.2 Programme compilieren und ausführen

2.3 Elementare Aus- und Eingabe

2.4 Elementares Rechnen

2.5 Verzweigungen

2.6 Schleifen

2.7 Seiteneffekte

2.8 Strukturierte Programmierung

...

## 3 Einführung in C++

...



Änderungen  
vorbehalten

## 2 Wiederholung: Programmieren in C

### 2.1 Hello, world!

Text ausgeben

```
#include <stdio.h>
```

```
int main (void)
```

```
{  
    printf ("Hello, _world!\n");  
    return 0;  
}
```

## 2.2 Programme compilieren und ausführen


```
$ gcc hello-1.c -o hello-1  
$ ./hello-1  
Hello, world!  
$
```

## 2.2 Programme compilieren und ausführen

```
$ gcc -Wall -O hello-1.c -o hello-1  
$ ./hello-1  
Hello, world!  
$
```

## 2.2 Programme compilieren und ausführen

```
$ gcc -Wall -O hello-1.c -o hello-1
$ ./hello-1
Hello, world!
$
```



-Wall	alle Warnungen einschalten
-O	optimieren
-O3	maximal optimieren
-Os	Codegröße optimieren
...	gcc hat <i>sehr viele</i> Optionen.

## 2.3 Elementare Aus- und Eingabe

Wert ausgeben

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("Die_Antwort_lautet:_");
```

```
    printf (42);
```

```
    printf ("\n");
```

```
    return 0;
```

```
}
```

## 2.3 Elementare Aus- und Eingabe

Wert ausgeben

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("Die_Antwort_lautet:_");
```

```
    printf (42);
```

```
    printf ("\n");
```

```
    return 0;
```

```
}
```

→ Absturz

## 2.3 Elementare Aus- und Eingabe

Wert ausgeben

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("Die_Antwort_lautet:_%d\n", 42);
```

```
    return 0;
```

```
}
```

Formatspezifikation „d“: „dezimal“





## 2.3 Elementare Aus- und Eingabe

Wert ausgeben

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("Die_Antwort_lautet:_%d\n", 42);
```

```
    return 0;
```

```
}
```



Formatspezifikation „d“: „dezimal“

Weitere Formatspezifikationen:  
siehe Online-Dokumentation  
(z. B. man 3 printf),  
Internet-Recherche oder Literatur

## 2.3 Elementare Aus- und Eingabe

Wert einlesen

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    double a;
```

```
    printf ("Bitte_eine_Zahl_eingeben:_");
```

```
    scanf ("%lf", &a);
```

```
    printf ("Ihre_Antwort_war:_%lf\n", a);
```

```
    return 0;
```

```
}
```

Formatspezifikation „lf“:  
„long floating-point“

Das „&“ nicht vergessen!

## 2.4 Elementares Rechnen

Wert an Variable zuweisen

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    int a;
```

```
    printf ("Bitte_eine_Zahl_eingeben:_");
```

```
    scanf ("%d", &a);
```

```
    a = 2 * a;
```

```
    printf ("Das_Doppelte_ist:_%d\n", a);
```

```
    return 0;
```

```
}
```

## 2.5 Verzweigungen

### if-Verzweigung

```
if (b != 0)  
    printf ("%d\n", a / b);
```

## 2.5 Verzweigungen

### if-Verzweigung

```
if (b != 0)
    printf ("%d\n", a / b);
```

### Wahrheitswerte in C: numerisch

0 steht für *falsch* (*false*),  
≠ 0 steht für *wahr* (*true*).

```
if (b)
    printf ("%d\n", a / b);
```

## 2.6 Schleifen

### while-Schleife

```
a = 1;  
while (a <= 10)  
{  
    printf ("%d\n", a);  
    a = a + 1;  
}
```

## 2.6 Schleifen

### **while**-Schleife

```
a = 1;
while (a <= 10)
{
    printf ("%d\n", a);
    a = a + 1;
}
```

### **for**-Schleife

```
for (a = 1; a <= 10; a = a + 1)
    printf ("%d\n", a);
```

## 2.6 Schleifen

### **while**-Schleife

```
a = 1;
while (a <= 10)
{
    printf ("%d\n", a);
    a = a + 1;
}
```

### **for**-Schleife

```
for (a = 1; a <= 10; a = a + 1)
    printf ("%d\n", a);
```

### **do-while**-Schleife

```
a = 1;
do
{
    printf ("%d\n", a);
    a = a + 1;
}
while (a <= 10);
```



## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)  
{  
    printf ("%d\n", 42);  
    "\n";  
    return 0;  
}
```

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("%d\n", 42);
```

```
    "\n";
```

← Ausdruck als Anweisung: Wert wird ignoriert

```
    return 0;
```

```
}
```

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    printf ("%d\n", 42);
```

```
    "\n";
```

```
    return 0;
```

```
}
```

← Ausdruck als Anweisung: Wert wird ignoriert

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

```
$ gcc -Wall -O side-effects-1.c -o side-effects-1
```

```
$ ./side-effects-1
```

```
42
```

```
3
```

```
$
```

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{
```

```
    int a = printf ("%d\n", 42);
```

```
    printf ("%d\n", a);
```

```
    return 0;
```

```
}
```

- `printf()` ist eine Funktion.

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

- `printf()` ist eine Funktion.
- „Haupteffekt“: Wert zurückliefern  
(hier: Anzahl der ausgegebenen Zeichen)

## 2.7 Seiteneffekte

```
#include <stdio.h>
```

```
int main (void)
```

```
{  
    int a = printf ("%d\n", 42);  
    printf ("%d\n", a);  
    return 0;  
}
```

- `printf()` ist eine Funktion.
- „Haupteffekt“: Wert zurückliefern  
(hier: Anzahl der ausgegebenen Zeichen)
- *Seiteneffekt*: Ausgabe



## 2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Inkrement: `foo++`
- Post-Dekrement: `foo—`
- Prä-Inkrement: `++foo`
- Prä-Dekrement: `—foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`
- Ignorieren: `,`

## 2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Inkrement: `foo++`
- Post-Dekrement: `foo—`
- Prä-Inkrement: `++foo`
- Prä-Dekrement: `—foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`
- Ignorieren: `,`

rot = mit Seiteneffekt

## 2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Inkrement: `foo++`
- Post-Dekrement: `foo—`
- Prä-Inkrement: `++foo`
- Prä-Dekrement: `—foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`
- Ignorieren: `,`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i++;
```

```
}
```

## 2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `!foo`
- Funktionsaufruf: `foo()`
- Post-Inkrement: `foo++`
- Post-Dekrement: `foo--`
- Prä-Inkrement: `++foo`
- Prä-Dekrement: `--foo`

Binäre Operatoren:

- Rechnen: `+` `-` `*` `/` `%`
- Vergleich: `==` `!=` `<` `>` `<=` `>=`
- Zuweisung: `=` `+=` `-=` `*=` `/=` `%=`
- Ignorieren: `,`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i++;
```

```
}
```

```
for (i = 0; i < 10; i++)
```

```
    printf ("%d\n", i);
```

## 2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Inkrement: `foo++`
- Post-Dekrement: `foo—`
- Prä-Inkrement: `++foo`
- Prä-Dekrement: `—foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`
- Ignorieren: `,`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
```

```
while (i < 10)
{
    printf ("%d\n", i);
    i++;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

```
i = 0;
```

```
while (i < 10)
    printf ("%d\n", i++);
```

## 2.7 Seiteneffekte bei Operatoren

Unäre Operatoren:

- Negation: `—foo`
- Funktionsaufruf: `foo ()`
- Post-Inkrement: `foo++`
- Post-Dekrement: `foo—`
- Prä-Inkrement: `++foo`
- Prä-Dekrement: `—foo`

Binäre Operatoren:

- Rechnen: `+ — * / %`
- Vergleich: `== != < > <= >=`
- Zuweisung: `= += -= *= /= %=`
- Ignorieren: `,`

rot = mit Seiteneffekt

```
int i;
```

```
i = 0;
```

```
while (i < 10)
{
    printf ("%d\n", i);
    i++;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

```
i = 0;
```

```
while (i < 10)
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

```
int i;
```

```
i = 0;
```

```
while (i < 10)
```

```
{
```

```
    printf ("%d\n", i);
```

```
    i++;
```

```
}
```

```
for (i = 0; i < 10; i++)
```

```
    printf ("%d\n", i);
```

```
i = 0;
```

```
while (i < 10)
```

```
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

```
i = 0;  
while (1)  
{  
    if (i >= 10)  
        break;  
    printf ("%d\n", i++);  
}
```

```
int i;
```

```
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i++;  
}
```

```
for (i = 0; i < 10; i++)  
    printf ("%d\n", i);
```

```
i = 0;  
while (i < 10)  
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```



```
i = 0;
while (1)
{
    if (i >= 10)
        break;
    printf ("%d\n", i++);
}
```

```
i = 0;
loop:
if (i >= 10)
    goto endloop;
printf ("%d\n", i++);
goto loop;
endloop:
```

```
int i;
```

```
i = 0;
while (i < 10)
{
    printf ("%d\n", i);
    i++;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

```
i = 0;
while (i < 10)
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

## 2.8 Strukturierte Programmierung

```
i = 0;
while (1)
{
    if (i >= 10)
        break;
    printf ("%d\n", i++);
}
```

```
i = 0;
loop:
if (i >= 10)
    goto endloop;
printf ("%d\n", i++);
goto loop;
endloop:
```

```
int i;

i = 0;
while (i < 10)
{
    printf ("%d\n", i);
    i++;
}
```

```
for (i = 0; i < 10; i++)
    printf ("%d\n", i);
```

```
i = 0;
while (i < 10)
    printf ("%d\n", i++);
```

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

## 2.8 Strukturierte Programmierung

```
i = 0;  
while (1) fragwürdig  
{  
    if (i >= 10)  
        break;  
    printf ("%d\n", i++);  
}
```

```
i = 0;  
loop:  
if (i >= 10) sehr fragwürdig  
    goto endloop;  
printf ("%d\n", i++);  
goto loop;  
endloop:
```

(siehe z. B.:  
<http://xkcd.com/292/>)

```
int i;  
  
i = 0;  
while (i < 10)  
{  
    printf ("%d\n", i);  
    i++;  
}
```

gut

```
for (i = 0; i < 10; i++)  
    printf ("%d\n", i);
```

```
i = 0;  
while (i < 10)  
    printf ("%d\n", i++);
```

nur, wenn  
Sie wissen,  
was Sie tun

```
for (i = 0; i < 10; printf ("%d\n", i++));
```

# Vertiefung Software-Entwicklung in C++

<https://gitlab.cvh-server.de/pgerwinski/cpp.git>

## 1 Einführung

1.1 Was ist C?

1.2 Was ist C++?

## 2 Wiederholung: Programmieren in C

2.1 Hello, world!

2.2 Programme compilieren und ausführen

2.3 Elementare Aus- und Eingabe

2.4 Elementares Rechnen

2.5 Verzweigungen

2.6 Schleifen

2.7 Seiteneffekte

2.8 Strukturierte Programmierung

...

## 3 Einführung in C++

...



Änderungen  
vorbehalten