

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

Wintersemester 2019/20

Wichtiger Hinweis

Diese Vortragsfolien dienen dazu, den Vortrag der/des Lehrenden zu unterstützen. Sie enthalten **nur einen Teil** der Lerninhalte. Wie groß dieser Teil ist, hängt von den konkreten Lerninhalten ab und kann von „praktisch alles“ bis „praktisch gar nichts“ schwanken. Diese Folien alleine sind daher **nicht für ein Selbststudium geeignet!** Hierfür sei auf das Skript verwiesen, in dem allerdings keine tagesaktuellen Änderungen enthalten sind.

Mindestens genauso wichtig wie die Vortragsfolien sind die Beispiel-Programme, die vor Ihren Augen in den Vorlesungen erarbeitet werden. Diese sind im Git-Repository (<https://gitlab.cvh-server.de/pgerwinski/hp.git>) mit allen Zwischenschritten enthalten und befinden sich in den zu den jeweiligen Kalenderdaten gehörenden Verzeichnissen (z. B. für den 10.10.2019 unter <https://gitlab.cvh-server.de/pgerwinski/hp/tree/master/20191010/>).

Wenn Sie die Übungsaufgaben bearbeiten, nutzen Sie die Gelegenheit, Ihre Lösungen in den Übungen überprüfen zu lassen. Wer nach Vergleich mit der Musterlösung zu dem Schluß kommt, alles richtig gelöst zu haben, erlebt sonst in der Klausur oft eine unangenehme Überraschung.

In jedem Fall: *Viel Erfolg!*

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

10. Oktober 2019

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

rerum naturalium = der natürlichen Dinge (lat.)

10. Oktober 2019



Hardwarenahe Programmierung

Man kann Computer vollständig beherrschen.

Hardwarenahe Programmierung

Man kann Computer vollständig beherrschen.

Programmierung in C

- Hardware direkt ansprechen und effizient einsetzen
- ... bis hin zu komplexen Software-Projekten
- Programmierkenntnisse werden nicht vorausgesetzt, aber schnelles Tempo

Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen

Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen

Hardware und/oder Betriebssystem



Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich

Hardware und/oder Betriebssystem



Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich

„High-Level-Assembler“

Hardware und/oder Betriebssystem



Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich

„High-Level-Assembler“

- kein „Fallschirm“

Hardware und/oder Betriebssystem



Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich

„High-Level-Assembler“

Hardware und/oder Betriebssystem

- kein „Fallschirm“
- kompakte Schreibweise

Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich

„High-Level-Assembler“

Hardware und/oder Betriebssystem

- kein „Fallschirm“
- kompakte Schreibweise

Unix-Hintergrund

Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich



„High-Level-Assembler“

Hardware und/oder Betriebssystem

- kein „Fallschirm“
- kompakte Schreibweise

Unix-Hintergrund

- Baukastenprinzip

Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich



„High-Level-Assembler“

Hardware und/oder Betriebssystem

- kein „Fallschirm“
- kompakte Schreibweise

Unix-Hintergrund

- Baukastenprinzip
- konsequente Regeln

Hardwarenahe Programmierung

Was ist C?

Etabliertes Profi-Werkzeug

- kleinster gemeinsamer Nenner für viele Plattformen
- leistungsfähig, aber gefährlich



„High-Level-Assembler“

Hardware und/oder Betriebssystem

- kein „Fallschirm“
- kompakte Schreibweise

Unix-Hintergrund

- Baukastenprinzip
- konsequente Regeln
- kein „Fallschirm“

Zu dieser Lehrveranstaltung



- **Lehrmaterialien:**

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

- **Klausur:**

Zeit: 150 Minuten

Zulässige Hilfsmittel:

- Schreibgerät
- beliebige Unterlagen in Papierform und/oder auf Datenträgern
- elektronische Rechner (Notebook, Taschenrechner o. ä.)
- *kein* Internet-Zugang

- **Übungen**

finden bereits diese Woche statt.

- **Praktikumstermine:**

- Versuch 1: 17. 10. und 24. 10. 2018
- Versuch 2 bis 4: Termine werden noch bekanntgegeben.

Hardwarenahe Programmierung

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

1 Einführung

1.1 Was ist hardwarenahe Programmierung?

1.2 Programmierung in C

1.3 Zu dieser Lehrveranstaltung

2 Einführung 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 Strukturierte Programmierung

...

3 Bibliotheken

...

2 Einführung in C

2.1 Hello, world!

Text ausgeben

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

```
int main (void)
```

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

2 Einführung in C

2.1 Hello, world!

Text ausgeben

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

```
int main (void)
```

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

printf = „print formatted“

2 Einführung in C

2.1 Hello, world!

Text ausgeben

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

```
int main (void)
```

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

printf = „print formatted“

\n: Zeilenschaltung



2.2 Programme compilieren und ausführen

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

2.2 Programme compilieren und ausführen

```
$ gcc hello-1.c -o hello-1
```

```
$ ./hello-1
```

```
Hello, world!
```

```
$
```

`-o hello-1`

Name für Ausgabe-Datei („output“)
unter Unix: ohne Endung
unter MS-Windows: Endung `.exe`

2.2 Programme compilieren und ausführen

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

Hier: Kommandozeilen-Interface (CLI)

- Der C-Compiler (hier: `gcc`) muß installiert sein und sich im `PATH` befinden.
- Der Quelltext (hier: `hello.c`) muß sich im aktuellen Verzeichnis befinden.
- aktuelles Verzeichnis herausfinden: `pwd`
- aktuelles Verzeichnis wechseln: `cd foobar`, `cd ..`
- Inhalt des aktuellen Verzeichnisses ausgeben: `ls`, `ls -l`
- Ausführen des Programms (`hello-1`) im aktuellen Verzeichnis (`.`):
`./hello-1`

Alternative: Integrierte Entwicklungsumgebung (IDE)
mit graphischer Benutzeroberfläche (GUI)

- Das können Sie bereits.

2.2 Programme compilieren und ausführen

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

GNU Compiler Collection (GCC) für verschiedene Plattformen:

- GNU/Linux: [gcc](#)
- Apple Mac OS: [Xcode](#)
- Microsoft Windows: [Cygwin](#)
oder [MinGW](#) mit [MSYS](#)
- außerdem: Texteditor
[vi\(m\)](#), [nano](#), [Emacs](#), [Notepad++](#), ...
(Microsoft Notepad ist *nicht* geeignet!)

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!
$
```



-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 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;
```

```
}
```

2.4 Elementares Rechnen

Variable bei Deklaration initialisieren

```
int a = 42;  
a = 137;
```

Achtung: Initialisierung \neq Zuweisung

Die beiden Gleichheitszeichen haben
subtil unterschiedliche Bedeutungen!

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);
```



```
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++));
```

2.7 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++));
```

2.7 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++));
```

Hardwarenahe Programmierung

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

1 Einführung

- 1.1 Was ist hardwarenahe Programmierung?
- 1.2 Programmierung in C
- 1.3 Zu dieser Lehrveranstaltung

2 Einführung 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 Strukturierte Programmierung
- 2.8 Seiteneffekte
- 2.9 Funktionen
- 2.10 Zeiger

...

3 Bibliotheken

...