

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

29. Oktober 2018

Hardwarenahe Programmierung

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

1 Einführung

2 Einführung in C

...

2.7 Seiteneffekte

2.8 Strukturierte Programmierung

2.9 Funktionen

2.10 Zeiger

2.11 Arrays und Strings

2.12 Strukturen

2.13 Dateien und Fehlerbehandlung

2.14 Parameter des Hauptprogramms

2.15 String-Operationen

3 Bibliotheken

...

2.10 Zeiger

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

```
void calc_answer (int *a)
{
    *a = 42;
}
```

- `*a` ist eine **int**.
- unärer Operator `*`:
Pointer-Derferenzierung

→ `a` ist ein Zeiger (Pointer) auf eine **int**.

```
int main (void)
{
    int answer;
    calc_answer (&answer);
    printf ("The_answer_is_%d.\n", answer);
    return 0;
}
```

- unärer Operator `&`: Adresse


2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime;  
    for (int i = 0; i < 5; i++)  
        printf ("%d\n", *(p + i));  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.


2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    int *p = prime;  
    for (int i = 0; i < 5; i++)  
        printf ("%d\n", p[i]);  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`. 
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise:
`p[i]` statt `*(p + i)`

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    for (int i = 0; i < 5; i++)  
        printf ("%d\n", prime[i]);  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`.
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise:
`p[i]` statt `*(p + i)`

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[5] = { 2, 3, 5, 7, 11 };  
    for (int *p = prime;  
         p < prime + 5; p++)  
        printf ("%d\n", *p);  
    return 0;  
}
```

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`.
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise:
`p[i]` statt `*(p + i)`
- Zeiger-Arithmetik:
`p++` rückt den Zeiger `p` um eine `int` weiter.

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[6] = { 2, 3, 5, 7, 11, 0 };  
    for (int *p = prime; *p; p++)  
        printf ("%d\n", *p);  
    return 0;  
}
```

Ende-Markierung



- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`.
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise: `p[i]` statt `*(p + i)`
- Zeiger-Arithmetik: `p++` rückt den Zeiger `p` um eine `int` weiter.

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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

```
int main (void)
```

```
{  
    int prime[] = { 2, 3, 5, 7, 11, 0 };  
    for (int *p = prime; *p; p++)  
        printf ("%d\n", *p);  
    return 0;  
}
```

Ende-Markierung



**Die Länge des Arrays
ist *nicht* veränderlich!**

- `prime` ist ein Array von fünf ganzen Zahlen.
- `prime` ist ein Zeiger auf eine `int`.
- `p + i` ist ein Zeiger auf den `i`-ten Nachbarn von `*p`.
- `*(p + i)` ist der `i`-te Nachbar von `*p`.
- Andere Schreibweise: `p[i]` statt `*(p + i)`
- Zeiger-Arithmetik: `p++` rückt den Zeiger `p` um eine `int` weiter.
- Array ohne explizite Längenangabe: Compiler zählt selbst

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

```
    char hello[] = "Hello, world!\n";
```

```
    for (char *p = hello; *p; p++)
```

```
        printf ("%d", *p);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Der letzte **char** muß 0 sein. Er kennzeichnet das Ende des Strings.

2.11 Arrays und Strings

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

```
int main (void)
{
    char hello[] = "Hello, world!\n";
    for (char *p = hello; *p; p++)
        printf ("%c", *p);
    return 0;
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Der letzte **char** muß 0 sein. Er kennzeichnet das Ende des Strings.
- Die Formatspezifikation entscheidet über die Ausgabe:
 - %d** dezimal
 - %c** Zeichen
 - %x** hexadezimal

2.11 Arrays und Strings

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

```
int main (void)
{
    char hello[] = "Hello, _world!\n";
    printf ("%s", hello);
    return 0;
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Der letzte **char** muß 0 sein. Er kennzeichnet das Ende des Strings.
- Die Formatspezifikation entscheidet über die Ausgabe:

%d	dezimal	%c	Zeichen
%x	hexadezimal	%s	String

2.11 Arrays und Strings

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

```
int main (void)
{
    char *hello = "Hello, _world!\n";
    printf ("%s", hello);
    return 0;
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**, also ein Zeiger auf **chars** also ein Zeiger auf (kleinere) Integer.
- Der letzte **char** muß 0 sein. Er kennzeichnet das Ende des Strings.
- Die Formatspezifikation entscheidet über die Ausgabe:

%d	dezimal	%c	Zeichen
%x	hexadezimal	%s	String

2.11 Arrays und Zeiger und Strings

h =

H	e	l	l	o	!	\n	Null
---	---	---	---	---	---	----	------

H	e	l	l	o	!	\n	Null
---	---	---	---	---	---	----	------

↑
h =

--

```
char h[] = "Hello!\n";
```

```
printf ("%s", h);
```

```
h[1] = 'a';
```

```
int i = 0;
```

```
while (h[i])  
    printf ("%c", h[i++]);
```

```
while (*h)  
    printf ("%c", *h++);
```

```
char *h = "Hello!\n";
```

```
printf ("%s", h);
```

```
h[1] = 'a';
```

```
int i = 0;
```

```
while (h[i])  
    printf ("%c", h[i++]);
```

```
while (*h)  
    printf ("%c", *h++);
```

2.11 Arrays und Zeiger und Strings

h =

H	e	l	l	o	!	\n	Null
---	---	---	---	---	---	----	------

h =

H	e	l	l	o	!	\n	Null
---	---	---	---	---	---	----	------

↑

--

 ohne Schreibzugriff

```
char h[] = "Hello!\n";
```

```
printf ("%s", h);
```

```
h[1] = 'a';
```

```
int i = 0;
```

```
while (h[i])  
    printf ("%c", h[i++]);
```

```
while (*h)  
    printf ("%c", *h++);
```

```
char *h = "Hello!\n";
```

```
printf ("%s", h);
```

```
h[1] = 'a';
```

```
int i = 0;
```

```
while (h[i])  
    printf ("%c", h[i++]);
```

```
while (*h)  
    printf ("%c", *h++);
```

2.11 Arrays und Zeiger und Strings

`h =`

H	e	l	l	o	!	\n	Null
---	---	---	---	---	---	----	------

Adresse des Arrays

```
char h[] = "Hello!\n";
```

```
printf ("%s", h);
```

```
h[1] = 'a';
```

```
int i = 0;
```

```
while (h[i])  
    printf ("%c", h[i++]);
```

```
while (*h)  
    printf ("%c", *h++);
```

H	e	l	l	o	!	\n	Null
---	---	---	---	---	---	----	------

`h =`

--

ohne Schreibzugriff

```
char *h = "Hello!\n";
```

```
printf ("%s", h);
```

```
h[1] = 'a';
```

```
int i = 0;
```

```
while (h[i])  
    printf ("%c", h[i++]);
```

```
while (*h)  
    printf ("%c", *h++);
```


2.11 Arrays und Strings und Zeichen

„Alles ist Zahl.“ – Schule der Pythagoreer, 6. Jh. v. Chr.

"Hello"		{ 72, 101, 108, 108, 111, 0 }
'H'	ist nur eine andere	72
	Schreibweise für	
'a' + 4		'e'

- Welchen Zahlenwert hat '*' im Zeichensatz?

```
printf ("%d\n", '*');
```

(normalerweise: ASCII)

- Ist **char** **ch** ein Großbuchstabe?

```
if (ch >= 'A' && ch <= 'Z')
```

...

- Groß- in Kleinbuchstaben umwandeln

```
ch += 'a' - 'A';
```

2.12 Strukturen

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

```
typedef struct
```

```
{  
    char day, month;  
    int year;  
}  
date;
```

- Eigener Datentyp: `date`
- zusammengesetzt aus elementarerer Datentypen

```
int main (void)
```

```
{  
    date today = { 30, 10, 2014 };  
    printf ("%d.%d.%d\n", today.day, today.month, today.year);  
    return 0;  
}
```

2.12 Strukturen

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

```
typedef struct
```

```
{
```

```
    char day, month;
```

```
    int year;
```

```
}
```

```
date;
```

```
void set_date (date *d)
```

```
{
```

```
    (*d).day = 30;
```

```
    (*d).month = 10;
```

```
    (*d).year = 2014;
```

```
}
```

- Eigener Datentyp: `date`
- zusammengesetzt aus elementarerer Datentypen
- `set_date()`: „Methode“ von `date`
- `d->day` ist Abkürzung für `(*d).day`

```
int main (void)
```

```
{
```

```
    date today;
```

```
    set_date (&today);
```

```
    printf ("%d.%d.%d\n", today.day,  
            today.month, today.year);
```

```
    return 0;
```

```
}
```

2.12 Strukturen

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

```
typedef struct
```

```
{
```

```
    char day, month;
```

```
    int year;
```

```
}
```

```
date;
```

```
void set_date (date *d)
```

```
{
```

```
    d->day = 30;
```

```
    d->month = 10;
```

```
    d->year = 2014;
```

```
}
```

- Eigener Datentyp: `date`
- zusammengesetzt aus elementarerer Datentypen
- `set_date()`: „Methode“ von `date`
- `d->day` ist Abkürzung für `(*d).day`

```
int main (void)
```

```
{
```

```
    date today;
```

```
    set_date (&today);
```

```
    printf ("%d.%d.%d\n", today.day,  
            today.month, today.year);
```

```
    return 0;
```

```
}
```

2.13 Dateien und Fehlerbehandlung

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

```
int main (void)
```

```
{
```

```
    FILE *f = fopen ("fhello.txt", "w");
```

```
    fprintf (f, "Hello, world!\n");
```

```
    fclose (f);
```

```
    return 0;
```

```
}
```

2.13 Dateien und Fehlerbehandlung

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

```
int main (void)
```

```
{
```

```
    FILE *f = fopen ("fhello.txt", "w");
```

```
    if (f)
```

```
    {
```

```
        fprintf (f, "Hello,_world!\n");
```

```
        fclose (f);
```

```
    }
```

```
    return 0;
```

```
}
```

- Wenn die Datei nicht geöffnet werden kann, gibt `fopen()` den Wert `NULL` zurück.

2.13 Dateien und Fehlerbehandlung

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

```
#include <errno.h>
```

```
int main (void)
```

```
{
```

```
    FILE *f = fopen ("fhello.txt", "w");
```

```
    if (f)
```

```
    {
```

```
        fprintf (f, "Hello,_world!\n");
```

```
        fclose (f);
```

```
    }
```

```
    else
```

```
        fprintf (stderr, "error_#%d\n", errno);
```

```
    return 0;
```

```
}
```

- Wenn die Datei nicht geöffnet werden kann, gibt `fopen()` den Wert `NULL` zurück.
- Die globale Variable `int errno` enthält dann die Nummer des Fehlers. Benötigt: `#include <errno.h>`

2.13 Dateien und Fehlerbehandlung

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

```
int main (void)
{
    FILE *f = fopen ("fhello.txt", "w");
    if (f)
    {
        fprintf (f, "Hello, _world!\n");
        fclose (f);
    }
    else
    {
        char *msg = strerror (errno);
        fprintf (stderr, "%s\n", msg);
    }
    return 0;
}
```

- Wenn die Datei nicht geöffnet werden kann, gibt `fopen()` den Wert `NULL` zurück.
- Die globale Variable `int errno` enthält dann die Nummer des Fehlers. Benötigt: `#include <errno.h>`
- Die Funktion `strerror()` wandelt `errno` in einen Fehlermeldungstext um. Benötigt: `#include <string.h>`

2.13 Dateien und Fehlerbehandlung

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

```
#include <errno.h>
```

```
#include <error.h>
```

```
int main (void)
```

```
{
```

```
    FILE *f = fopen ("fhello.txt", "w");
```

```
    if (!f)
```

```
        error (-1, errno, "cannot_open_file");
```

```
    fprintf (f, "Hello,_world!\n");
```

```
    fclose (f);
```

```
    return 0;
```

```
}
```

- Wenn die Datei nicht geöffnet werden kann, gibt `fopen()` den Wert `NULL` zurück.
- Die globale Variable `int errno` enthält dann die Nummer des Fehlers. Benötigt: `#include <errno.h>`
- Die Funktion `strerror()` wandelt `errno` in einen Fehlermeldungstext um. Benötigt: `#include <string.h>`
- Die Funktion `error()` gibt eine Fehlermeldung aus und beendet das Programm. Benötigt: `#include <error.h>`

2.13 Dateien und Fehlerbehandlung

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

```
int main (void)
```

```
{
    FILE *f = fopen ("fhello.txt", "w");
    if (!f)
        error (-1, errno, "cannot_open_file");
    fprintf (f, "Hello,_world!\n");
    fclose (f);
    return 0;
}
```

- Wenn die Datei nicht geöffnet werden kann, gibt `fopen()` den Wert `NULL` zurück.
- Die globale Variable `int errno` enthält dann die Nummer des Fehlers. Benötigt: `#include <errno.h>`
- Die Funktion `strerror()` wandelt `errno` in einen Fehlermeldungstext um. Benötigt: `#include <string.h>`
- Die Funktion `error()` gibt eine Fehlermeldung aus und beendet das Programm. Benötigt: `#include <error.h>`
- **Niemals Fehler einfach ignorieren!**

2.14 Parameter des Hauptprogramms

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

```
int main (int argc, char **argv)
{
    printf ("argc=%d\n", argc);
    for (int i = 0; i < argc; i++)
        printf ("argv[%d]=\n", i, argv[i]);
    return 0;
}
```

2.14 Parameter des Hauptprogramms

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

```
int main (int argc, char **argv)
{
    printf ("argc=%d\n", argc);
    for (int i = 0; *argv; i++, argv++)
        printf ("argv[%d]=\n", i, *argv);
    return 0;
}
```

Hardwarenahe Programmierung

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

1 Einführung

2 Einführung in C

...

2.10 Zeiger

2.11 Arrays und Strings

2.12 Strukturen

2.13 Dateien und Fehlerbehandlung

2.14 Parameter des Hauptprogramms

2.15 String-Operationen

3 Bibliotheken

3.1 Der Präprozessor

3.2 Bibliotheken einbinden

3.3 Bibliotheken verwenden

...

...