

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

24. Oktober 2022

Hardwarenahe Programmierung

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

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

3.4 Projekt organisieren: make

4 Hardwarenahe Programmierung

5 Algorithmen

...

2.12 Strukturen

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

```
typedef struct
```

```
{
```

```
    char day, month;
```

```
    int year;
```

```
}
```

```
date;
```

```
int main (void)
```

```
{
```

```
    date today = { 24, 10, 2022 };
```

```
    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 = 24;
```

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

```
    (*d).year = 2022;
```

```
}
```

```
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>
```

foo->bar ist Abkürzung für (*foo).bar

```
typedef struct
```

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

```
date;
```

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

```
{  
    d->day = 24;  
    d->month = 10;  
    d->year = 2022;  
}
```

```
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 = 24;  
    d->month = 10;  
    d->year = 2022;  
}
```

`foo->bar` ist Abkürzung für `(*foo).bar`

Eine Funktion, die mit einem **struct** arbeitet, kann man eine *Methode* des **struct** nennen.

```
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 (errno, 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 (errno, 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]=" "%s"\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;
```

```
}
```

2.15 String-Operationen

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

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

```
int main (void)
```

```
{
```

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

```
    printf ("%s\n", hello);
```

```
    printf ("%zd\n", strlen (hello));
```

```
    printf ("%s\n", hello + 7);
```

```
    printf ("%zd\n", strlen (hello + 7));
```

```
    hello[5] = 0;
```

```
    printf ("%s\n", hello);
```

```
    printf ("%zd\n", strlen (hello));
```

```
    return 0;
```

```
}
```

2.15 String-Operationen

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

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

```
int main (void)
```

```
{
```

```
    char *anton = "Anton";
```

```
    char *zacharias = "Zacharias";
```

```
    printf ("%d\n", strcmp (anton, zacharias));
```

```
    printf ("%d\n", strcmp (zacharias, anton));
```

```
    printf ("%d\n", strcmp (anton, anton));
```

```
    char buffer[100] = "Huber_";
```

```
    strcat (buffer, anton);
```

```
    printf ("%s\n", buffer);
```

```
    return 0;
```

```
}
```


2.15 String-Operationen

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

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

```
int main (void)
```

```
{
```

```
    char *anton = "Anton";
```

```
    char *huber = "Huber";
```

```
    int size = strlen (huber) + strlen (" ") + strlen (anton) + 1;
```

```
    char buffer[size];
```

```
    buffer[0] = 0;
```

```
    strcat (buffer, huber);
```

```
    strcat (buffer, " ");
```

```
    strcat (buffer, anton);
```

```
    printf ("%s\n", buffer);
```

```
    return 0;
```

```
}
```

2.15 String-Operationen

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

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

```
int main (void)
```

```
{
```

```
    char buffer[100] = "";
```

```
    sprintf (buffer, "Die_Antwort_lautet:_%d", 42);
```

```
    printf ("%s\n", buffer);
```

```
    char *answer = strstr (buffer, "Antwort");
```

```
    printf ("%s\n", answer);
```

```
    printf ("found_at:_%zd\n", answer - buffer);
```

```
    return 0;
```

```
}
```

2.15 String-Operationen

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

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

```
int main (void)
```

```
{
```

```
    char buffer[100] = "";
```

```
    snprintf (buffer, 100, "Die_Antwort_lautet:_%d", 42);
```

```
    printf ("%s\n", buffer);
```

```
    char *answer = strstr (buffer, "Antwort");
```

```
    printf ("%s\n", answer);
```

```
    printf ("found_at:_%zd\n", answer - buffer);
```

```
    return 0;
```

```
}
```

2 Einführung in C

Sprachelemente weitgehend komplett

Es fehlen:

- Ergänzungen (z. B. ternärer Operator, **union**, **unsigned**, **volatile**)
- Bibliotheksfunktionen (z. B. **malloc()**)

—> werden eingeführt, wenn wir sie brauchen

- Konzepte (z. B. rekursive Datenstrukturen, Klassen selbst bauen)

—> werden eingeführt, wenn wir sie brauchen, oder:

—> Literatur

(z. B. Wikibooks: C-Programmierung,
Dokumentation zu Compiler und Bibliotheken)

- Praxiserfahrung

—> Übung und Praktikum: nur Einstieg

—> selbständig arbeiten

Hardwarenahe Programmierung

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

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

3.4 Projekt organisieren: make

4 Hardwarenahe Programmierung

5 Algorithmen

...

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include <stdio.h>**: Standard-Verzeichnisse – Standard-Header
- **#include "answer.h"**: auch aktuelles Verzeichnis – eigene Header

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include <stdio.h>**: Standard-Verzeichnisse – Standard-Header
- **#include "answer.h"**: auch aktuelles Verzeichnis – eigene Header

#define SIX 6: Text ersetzen lassen – Konstante definieren

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include <stdio.h>**: Standard-Verzeichnisse – Standard-Header
- **#include "answer.h"**: auch aktuelles Verzeichnis – eigene Header

#define SIX 6: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

#define SIX 6: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!
- Berechnungen in Klammern setzen:
#define SIX (1 + 5)

3 Bibliotheken

3.1 Der Präprozessor

#include: Text einbinden

- **#include** <stdio.h>: Standard-Verzeichnisse – Standard-Header
- **#include** "answer.h": auch aktuelles Verzeichnis – eigene Header

#define SIX 6: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!
- Berechnungen in Klammern setzen:
#define SIX (1 + 5)
- Konvention: Großbuchstaben

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

extern int answer (**void**);

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

Funktion wird „anderswo“ definiert

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

extern int answer (**void**);

extern int printf (__const **char** *__restrict __format, ...);

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

extern int answer (**void**);

extern int printf (__const **char** *__restrict __format, ...);

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an *gcc* übergeben
- Zusammenfügen zu ausführbarem Programm durch den *Linker*

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

extern int answer (**void**);

extern int printf (__const **char** *__restrict __format, ...);

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben
- Zusammenfügen zu ausführbarem Programm durch den *Linker*
- vorcompilierte Bibliothek: `-lfoo`

3 Bibliotheken

3.2 Bibliotheken einbinden

Inhalt der Header-Datei: externe Deklarationen

```
extern int answer (void);
```

```
extern int printf (__const char *__restrict __format, ...);
```

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben
- Zusammenfügen zu ausführbarem Programm durch den *Linker*
- vorcompilierte Bibliothek: `-lfoo`
= Datei `libfoo.a` in Standard-Verzeichnis

3.3 Bibliothek verwenden (Beispiel: GTK+)

- **#include** <gtk/gtk.h>

3.3 Bibliothek verwenden (Beispiel: GTK+)

- `#include <gtk/gtk.h>`
- Mit `pkg-config --cflags --libs` erfährt man, welche Optionen und Bibliotheken man an `gcc` übergeben muß:

3.3 Bibliothek verwenden (Beispiel: GTK+)

- `#include <gtk/gtk.h>`

- Mit `pkg-config --cflags --libs` erfährt man, welche Optionen und Bibliotheken man an `gcc` übergeben muß:

```
$ pkg-config --cflags --libs gtk+-3.0
-pthread -I/usr/include/gtk-3.0 -I/usr/include/at-spi2-atk/2.0 -I/usr/include/at-spi-2.0 -I/usr/include/dbus-1.0 -I/usr/lib/x86_64-linux-gnu/dbus-1.0/include -I/usr/include/gtk-3.0 -I/usr/include/gio-unix-2.0/ -I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include/harfbuzz -I/usr/include/pango-1.0 -I/usr/include/atk-1.0 -I/usr/include/cairo -I/usr/include/pixman-1 -I/usr/include/freetype2 -I/usr/include/libpng16 -I/usr/include/gdk-pixbuf-2.0 -I/usr/include/libpng16 -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include -lgtk-3 -lgdk-3 -lpangocairo-1.0 -lpango-1.0 -latk-1.0 -lcairo-gobject -lcairo -lgdk_pixbuf-2.0 -lgio-2.0 -lgobject-2.0 -lglib-2.0
```

3.3 Bibliothek verwenden (Beispiel: GTK+)

- `#include <gtk/gtk.h>`
- Mit `pkg-config --cflags --libs` erfährt man, welche Optionen und Bibliotheken man an `gcc` übergeben muß.

→ Compiler-Aufruf:

```
$ gcc -Wall -O hello-gtk.c -pthread -I/usr/include/gtk-3.0 -I/usr/include/at-spi2-atk/2.0 -I/usr/include/at-spi-2.0 -I/usr/include/dbus-1.0 -I/usr/lib/x86_64-linux-gnu/dbus-1.0/include -I/usr/include/gtk-3.0 -I/usr/include/gio-unix-2.0/ -I/usr/include/cairo -I/usr/include/pango-1.0 -I/usr/include/harfbuzz -I/usr/include/pango-1.0 -I/usr/include/atk-1.0 -I/usr/include/cairo -I/usr/include/pixman-1 -I/usr/include/freetype2 -I/usr/include/libpng16 -I/usr/include/gdk-pixbuf-2.0 -I/usr/include/libpng16 -I/usr/include/glib-2.0 -I/usr/lib/x86_64-linux-gnu/glib-2.0/include -lgtk-3 -lgdk-3 -lpangocairo-1.0 -lpango-1.0 -latk-1.0 -lcairo-gobject -lcairo -lgdk_pixbuf-2.0 -lgio-2.0 -lgobject-2.0 -lglib-2.0 -o hello-gtk
```


3.3 Bibliothek verwenden (Beispiel: GTK+)

- `#include <gtk/gtk.h>`
- Mit `pkg-config --cflags --libs` erfährt man, welche Optionen und Bibliotheken man an `gcc` übergeben muß.

→ Compiler-Aufruf:

```
$ gcc -Wall -O hello-gtk.c $(pkg-config --cflags --libs  
    gtk+-3.0) -o hello-gtk
```

Optionen:

u. a. viele Include-Verzeichnisse:

`-I/usr/include/gtk-3.0`

Bibliotheken:

u. a. `-lgtk-3 -lcairo`

3.3 Bibliothek verwenden (Beispiel: GTK+)

- `#include <gtk/gtk.h>`
- Mit `pkg-config --cflags --libs` erfährt man, welche Optionen und Bibliotheken man an `gcc` übergeben muß.

→ Compiler-Aufruf:

```
$ gcc -Wall -O hello-gtk.c $(pkg-config --cflags --libs  
    gtk+-3.0) -o hello-gtk
```

- Auf manchen Plattformen kommt es auf die Reihenfolge an:

```
$ gcc -Wall -O $(pkg-config --cflags gtk+-3.0) \  
    hello-gtk.c $(pkg-config --libs gtk+-3.0) \  
    -o hello-gtk
```

(Backslash = „Es geht in der nächsten Zeile weiter.“)

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean draw (GtkWidget *widget, cairo_t *c, gpointer data)
```

```
{  
    /* Zeichenbefehle */  
    ...
```

```
    return FALSE;  
}
```

```
...
```

```
g_signal_connect (drawing_area, "draw", G_CALLBACK (draw), NULL);
```


→ GTK+ ruft immer dann, wenn es etwas zu zeichnen gibt,
die Funktion `draw` auf.

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean draw (GtkWidget *widget, cairo_t *c, gpointer data)
{
    /* Zeichenbefehle */
    ...

    return FALSE;
}
...
```



repräsentiert den
Bildschirm, auf den
gezeichnet werden soll

```
g_signal_connect (drawing_area, "draw", G_CALLBACK (draw), NULL);
```

→ GTK+ ruft immer dann, wenn es etwas zu zeichnen gibt,
die Funktion `draw` auf.

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean draw (GtkWidget *widget, cairo_t *c, gpointer data)
```

```
{  
    /* Zeichenbefehle */  
    ...
```

```
    return FALSE;  
}
```

```
...
```

```
g_signal_connect (drawing_area, "draw", G_CALLBACK (draw), NULL);
```

repräsentiert den
Bildschirm, auf den
gezeichnet werden soll

optionale Zusatzinformationen
für draw(), typischerweise
ein Zeiger auf ein struct

→ GTK+ ruft immer dann, wenn es etwas zu zeichnen gibt,
die Funktion `draw` auf.

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean timer (GtkWidget *widget)
{
    /* Rechenbefehle */
    ...

    gtk_widget_queue_draw_area (widget, 0, 0, WIDTH, HEIGHT);
    g_timeout_add (50, (GSourceFunc) timer, widget);
    return FALSE;
}

...

g_timeout_add (50, (GSourceFunc) timer, drawing_area);
```

→ GTK+ ruft nach 50 Millisekunden die Funktion **timer** auf.

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean timer (GtkWidget *widget)
```

```
{
```

```
    /* Rechenbefehle */
```

```
    ...
```

```
    gtk_widget_queue_draw_area (widget, 0, 0, WIDTH, HEIGHT);
```

```
    g_timeout_add (50, (GSourceFunc) timer, widget);
```

```
    return FALSE;
```

```
}
```

```
...
```

```
g_timeout_add (50, (GSourceFunc) timer, drawing_area);
```

Dieser Bereich soll
neu gezeichnet werden.



→ GTK+ ruft nach 50 Millisekunden die Funktion *timer* auf.

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean timer (GtkWidget *widget)
```

```
{
```

```
    /* Rechenbefehle */
```

```
    ...
```


```
    gtk_widget_queue_draw_area (widget, 0, 0, WIDTH, HEIGHT);
```

```
    g_timeout_add (50, (GSourceFunc) timer, widget);
```

```
    return FALSE;
```

```
}
```

Dieser Bereich soll
neu gezeichnet werden.



In weiteren 50 Millisekunden soll

... die Funktion erneut aufgerufen werden.

```
g_timeout_add (50, (GSourceFunc) timer, drawing_area);
```

→ GTK+ ruft nach 50 Millisekunden die Funktion *timer* auf.

3.3 Bibliothek verwenden (Beispiel: GTK+)

Selbst geschriebene Funktion übergeben: *Callback*

```
gboolean timer (GtkWidget *widget)
```

```
{
```

```
    /* Rechenbefehle */
```

```
    ...
```


```
    gtk_widget_queue_draw_area (widget, 0, 0, WIDTH, HEIGHT);
```

```
    g_timeout_add (50, (GSourceFunc) timer, widget);
```

```
    return FALSE;
```

```
}
```

Dieser Bereich soll
neu gezeichnet werden.



In weiteren 50 Millisekunden soll
die Funktion erneut aufgerufen werden.



Explizite Typumwandlung
eines Zeigers (später)



```
g_timeout_add (50, (GSourceFunc) timer, drawing_area);
```

→ GTK+ ruft nach 50 Millisekunden die Funktion **timer** auf.

Hardwarenahe Programmierung

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

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

3.4 Projekt organisieren: make

4 Hardwarenahe Programmierung

5 Algorithmen

...