

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

5. November 2018

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

...

...

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>`
- **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; i < argc; i++)
        printf ("argv[%d]=\n", i, argv[i]);
    return 0;
}
```

- **argc**: Anzahl der Parameter
- **argv**: Array von Strings
(= Zeiger auf Zeiger auf **chars**)
mit den Parametern
- Parameter Nr. 0:
Name des Programms selbst,
wie es aufgerufen wurde

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]_=_\"%s\"\n", i, *argv);
    return 0;
}
```

- **argc**: Anzahl der Parameter
- **argv**: Array von Strings
(= Zeiger auf Zeiger auf **chars**)
mit den Parametern
- Parameter Nr. 0:
Name des Programms selbst,
wie es aufgerufen wurde
- letzter Eintrag im Array:
NULL-Zeiger
(als Ende-Markierung)

2.14 Parameter des Hauptprogramms

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

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

- **argc**: Anzahl der Parameter
- **argv**: Array von Strings
(= Zeiger auf Zeiger auf **chars**)
mit den Parametern
- Parameter Nr. 0:
Name des Programms selbst,
wie es aufgerufen wurde
- letzter Eintrag im Array:
NULL-Zeiger
(als Ende-Markierung)

2.14 Parameter des Hauptprogramms

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

```
int main (int argc, char **argv)
```

```
{
```

```
    printf ("argc=_=%d\n", argc);
```

```
    int i = 0;
```

```
    while (*argv)
```

```
    {
```

```
        printf ("argv[%d]_=%s\n", i, *argv);
```

```
        i++;
```

```
        argv++;
```

```
    }
```

```
    return 0;
```

```
}
```

- **argc**: Anzahl der Parameter
- **argv**: Array von Strings
(= Zeiger auf Zeiger auf **chars**)
mit den Parametern
- Parameter Nr. 0:
Name des Programms selbst,
wie es aufgerufen wurde
- letzter Eintrag im Array:
NULL-Zeiger
(als Ende-Markierung)

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

...

...

2.15 String-Operationen

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

```
int main (void)
```

```
{
```

```
    char name[100];
```

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

```
    fgets (name, 100, stdin);
```

```
    printf ("Hallo,_%s!\n", name);
```

```
    return 0;
```

```
}
```

Probleme mit `scanf ("%s", name)`:

- Leerzeichen beendet Eingabe
- keine Prüfung der Puffergröße

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 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;
```

```
}
```

- `strlen()` gibt die Länge eines Strings zurück.
- Es enthält eine Schleife.

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;
```

```
}
```

- `strlen()` gibt die Länge eines Strings zurück.
- Es enthält eine Schleife.
- Typ des Rückgabewerts: `size_t` = ganze Zahl von der Größe eines Zeigers
- in `printf()`: `%zd` (*size*)

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;
```

```
}
```

- `strlen()` gibt die Länge eines Strings zurück.
- Es enthält eine Schleife.
- Typ des Rückgabewerts: `size_t` = ganze Zahl von der Größe eines Zeigers
- in `printf()`: `%zd` (*size*)
- Zeiger erhöhen: String vorne abschneiden

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;
```

```
}
```

- `strlen()` gibt die Länge eines Strings zurück.
- Es enthält eine Schleife.
- Typ des Rückgabewerts: `size_t` = ganze Zahl von der Größe eines Zeigers
- in `printf()`: `%zd` (*size*)
- Zeiger erhöhen: String vorne abschneiden
- 0-Symbol (= Ende-Markierung) in den String schreiben: String hinten abschneiden

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;
```

```
}
```

- `strlen()` gibt die Länge eines Strings zurück.
- Es enthält eine Schleife.
- Typ des Rückgabewerts: `size_t` = ganze Zahl von der Größe eines Zeigers
- in `printf()`: `%zd` (*size*)
- Zeiger erhöhen: String vorne abschneiden
- 0-Symbol (= Ende-Markierung) in den String schreiben: String hinten abschneiden
- **Der für den String reservierte Speicherplatz bleibt derselbe!**

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 *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;
```

```
}
```

- **strcmp()**: Strings vergleichen
- alphabetisch nach ASCII (Groß- < Kleinbuchstaben, ohne Umlaute usw.)
- Rückgabewert:
 - 1, wenn linker String kleiner,
 - +1, wenn rechter String kleiner,
 - 0, wenn beide Strings gleich

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;
```

```
}
```

- **strcmp()**: Strings vergleichen
- alphabetisch nach ASCII
(Groß- < Kleinbuchstaben,
ohne Umlaute usw.)
- Rückgabewert:
 - 1, wenn linker String kleiner,
 - +1, wenn rechter String kleiner,
 - 0, wenn beide Strings gleich
- **strcat()**: String anhängen
(*concatenate*)

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;
```

```
}
```

- **strcmp()**: Strings vergleichen
- alphabetisch nach ASCII (Groß- < Kleinbuchstaben, ohne Umlaute usw.)
- Rückgabewert:
 - 1, wenn linker String kleiner,
 - +1, wenn rechter String kleiner,
 - 0, wenn beide Strings gleich
- **strcat()**: String anhängen (*concatenate*)
- **Ob der Speicherplatz reicht, wird nicht geprüft!**

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>

- **sprintf()**: in einen String schreiben

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] = "";
```

```
    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;
```

```
}
```

- `sprintf()`: in einen String schreiben
- **Ob der Speicherplatz reicht, wird nicht geprüft!**

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;
```

```
}
```

- **sprintf()**: in einen String schreiben
- **Ob der Speicherplatz reicht, wird nicht geprüft!**
- **strstr()**: String in String suchen
- Rückgabewert: Zeiger auf den gefundenen String
- Index berechnen:
Zeiger – Zeiger = Zahl
von der Größe eines Zeigers

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.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

...

...