

Angewandte Informatik

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

6. November 2017

Angewandte Informatik

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

...

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.

2.11 Arrays und Strings

Ein Zeiger zeigt auf eine Variable und deren Nachbarn.

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;
```

```
}
```

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 eine Ansammlung von fünf ganzen Zahlen.

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.

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`.

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`.

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

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

```
int main (void)
{
    char hello_world[] = "Hello,_world!\n";
    int i = 0;
    while (hello_world[i] != 0)
        printf ("%d", hello_world[i++]);
    return 0;
}
```

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    int i = 0;
```

```
    while (hello_world[i])
```

```
        printf ("%d", hello_world[i++]);
```

```
    return 0;
```

```
}
```

2.11 Arrays und Strings

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

```
int main (void)
{
    char hello_world[] = "Hello,_world!\n";
    int i = 0;
    while (hello_world[i])
        printf ("%c", hello_world[i++]);
    return 0;
}
```


2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*p)
```

```
        printf ("%c", *p++);
```

```
    return 0;
```

```
}
```

- Ein **char** ist eine kleinere **int**.
- Ein „String“ in C ist ein Array von **chars**.

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*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**.

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*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.

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*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.

2.11 Arrays und Strings

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

```
int main (void)
```

```
{
```

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

```
    char *p = hello_world;
```

```
    while (*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_world[] = "Hello,_world!\n";
```

```
    char *p = hello_world;
```

```
    while (*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	%s	String

2.12 Strukturen

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

```
typedef struct
```

```
{
```

```
    char day, month;
```

```
    int year;
```

```
}
```

```
date;
```

```
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;
```

```
}
```

```
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;  
}
```

foo->bar ist Abkürzung für (*foo).bar

```
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%s\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 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 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

Angewandte Informatik

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

...

...

3 Bibliotheken

3.1 Der Präprozessor

`#include`

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 VIER 4: 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 VIER 4: 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 VIER 4: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!
- Berechnungen in Klammern setzen:
#define VIER (2 + 2)

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 VIER 4: Text ersetzen lassen – Konstante definieren

- Kein Semikolon!
- Berechnungen in Klammern setzen:
#define VIER (2 + 2)
- 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: OpenGL)

- Include-Dateien:

```
#include <GL/gl.h>
```

```
#include <GL/glu.h>
```

```
#include <GL/glut.h>
```

- Compiler-Aufruf:

```
gcc -Wall -O cube-1.c opengl-magic.c -lGL -lGLU -lglut \  
-o cube-1
```

3.3 Bibliothek verwenden (Beispiel: OpenGL)

Selbst geschriebene Funktion übergeben: *Callback*

```
void draw (void)
```

```
{
```

```
    ...
```

```
}
```

```
...
```

```
glutDisplayFunc (draw);
```

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

3.3 Bibliothek verwenden (Beispiel: OpenGL)

Selbst geschriebene Funktion übergeben: *Callback*

```
void key_handler (unsigned char key, int x, int y)
```

```
{
```

```
    ...
```

```
}
```

```
...
```

```
glutKeyboardFunc (key_handler);
```

→ OpenGL ruft immer dann, wenn eine Taste gedrückt wurde, die Funktion `key_handler` auf.

3.3 Bibliothek verwenden (Beispiel: OpenGL)

Selbst geschriebene Funktion übergeben: *Callback*

```
void key_handler (unsigned char key, int x, int y)
```

```
{
```

```
    ...
```

```
}
```

```
...
```

gedrückte Taste

Mausposition

```
glutKeyboardFunc (key_handler);
```

→ OpenGL ruft immer dann, wenn eine Taste gedrückt wurde, die Funktion `key_handler` auf.

5 Algorithmen

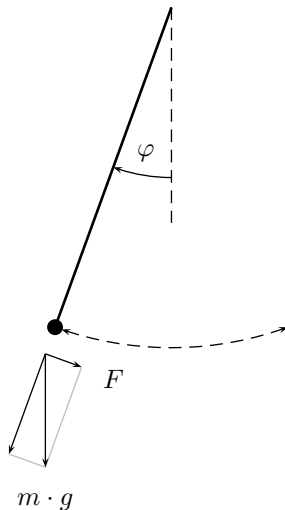
5.1 Differentialgleichungen

$$\varphi'(t) = \omega(t)$$

$$\omega'(t) = -\frac{g}{l} \cdot \sin \varphi(t)$$

- Von Hand (analytisch):
Lösung raten (Ansatz), Parameter berechnen
- Mit Computer (numerisch):
Eulersches Polygonzugverfahren

```
phi += dt * omega;  
omega += - dt * g / l * sin (phi);
```



5 Algorithmen

5.1 Differentialgleichungen

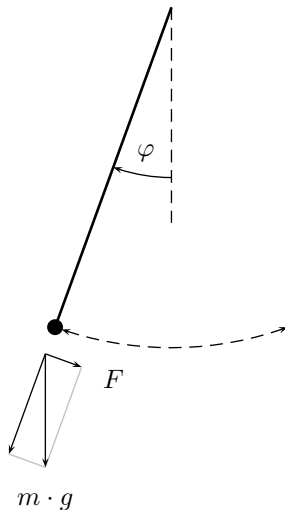
$$\varphi'(t) = \omega(t)$$

$$\omega'(t) = -\frac{g}{l} \cdot \sin \varphi(t)$$

- Von Hand (analytisch):
Lösung raten (Ansatz), Parameter berechnen
- Mit Computer (numerisch):
Eulersches Polygonzugverfahren

```
phi += dt * omega;  
omega += - dt * g / l * sin (phi);
```

Praktikumsaufgabe: Basketball



Angewandte Informatik

Hardwarenahe Programmierung

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

1 Einführung

2 Einführung in C

...

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

5.1 Differentialgleichungen

...

...