

Angewandte Informatik

Prof. Dr. rer. nat. Peter Gerwinski

19. November 2015

Sie können diese Vortragsfolien einschließlich Beispielprogramme, Skript und sonstiger Lehrmaterialien unter

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

herunterladen.

- **Files**
einzelne Dateien herunterladen
- **Download zip/tar.gz/tar.bz2/tar**
als Archiv herunterladen
- <https://gitlab.cvh-server.de/pgerwinski/ainf.git>
mit GIT herunterladen und synchronisieren

Sie können diese Vortragsfolien einschließlich Beispielprogramme, Skript und sonstiger Lehrmaterialien unter

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

herunterladen.

Achtung: Einige Dateien sind *Symlinks* – symbolische Verknüpfungen!

- Verweis auf eine andere Datei
- hier: Verweis auf gemeinsame Datei im Verzeichnis [common](#)
- Web-Interface zeigt Dateinamen als Text (statt Dateiinhalt)
- Beispiel: [earth-texture.rgb](#)
 - Textdatei mit Inhalt „[../common/earth-texture.rgb](#)“
 - bei Verwendung: keine Fehlermeldung, aber keine Textur
- Beim Herunterladen als Archiv-Datei (zip/tar.gz/tar.bz2/tar) sind die Symlinks darin korrekt gespeichert.
- ebenfalls korrekt:
[git clone https://gitlab.cvh-server.de/pgerwinski/ainf.git](#)

Ergänzungen

OpenGL

- **Doppelte Pufferung**

2 „Bildschirme“: einer zum Zeichnen; einer wird angezeigt

`opengl-magic-double.c`, `gluSwapBuffers()`

Ergänzungen

OpenGL

- **Doppelte Pufferung**

2 „Bildschirme“: einer zum Zeichnen; einer wird angezeigt
[opengl-magic-double.c](#), [gluSwapBuffers\(\)](#)

- **Im Display-Handler ([draw\(\)](#)) wirklich nur zeichnen!**

Wir haben nicht unter Kontrolle, wann, wie oft oder ob überhaupt diese Funktion aufgerufen wird.

Ergänzungen

Umgang mit Bibliotheken

- **Separates Compilieren**

```
$ gcc -c -Wall opengl-magic.c
```

```
$ gcc -c -Wall textured-spheres.c
```

```
$ gcc -Wall orbit-x.c -lGL -lGLU -lglut \  
    opengl-magic.o textured-spheres.o -o orbit-x
```

Ergänzungen

Umgang mit Bibliotheken

- **Separates Compilieren**

```
$ gcc -c -Wall opengl-magic.c
$ gcc -c -Wall textured-spheres.c
$ gcc -Wall orbit-x.c -lGL -lGLU -lglut \
    opengl-magic.o textured-spheres.o -o orbit-x
```

- Die **Link-Reihenfolge** kann eine Rolle spielen!

```
$ gcc -Wall orbit-x.c \
    opengl-magic.o textured-spheres.o \
    -lGL -lGLU -lglut -o orbit-x
```

Notfalls:

```
$ gcc -Wall orbit-x.c -lGL -lGLU -lglut \
    opengl-magic.o textured-spheres.o \
    -lGL -lGLU -lglut -o orbit-x
```

Ergänzungen

Parameter des Hauptprogramms

```
int main (int argc, char **argv)
{
    init_opengl (&argc, argv, "Orbit");
    ...
    return 0;
}
```


Ergänzungen

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;
}
```

Angewandte Informatik

1 Einführung

2 Einführung in C

3 Bibliotheken

3.1 Der Präprozessor

3.2 Bibliotheken einbinden

3.3 Bibliothek verwenden (Beispiel: OpenGL)

3.4 Projekt organisieren: make

4 Algorithmen

4.1 Differentialgleichungen

4.2 Rekursion

4.3 Stack und FIFO

4.4 Aufwandsabschätzungen

5 Hardwarenahe Programmierung

...

4 Algorithmen

4.1 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: Umlaufbahn

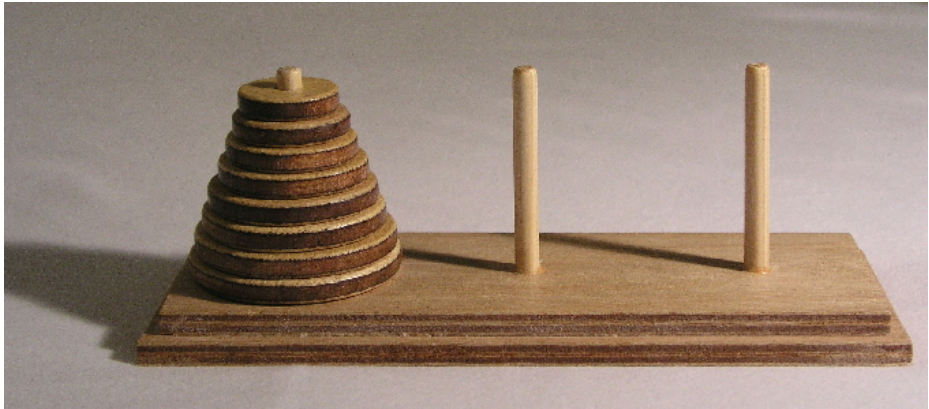
4.2 Rekursion

Vollständige Induktion: $\left. \begin{array}{l} \text{Aussage gilt für } n = 1 \\ \text{Schluß von } n - 1 \text{ auf } n \end{array} \right\} \text{ Aussage gilt für alle } n \in \mathbb{N}$

4.2 Rekursion

Vollständige Induktion: $\left. \begin{array}{l} \text{Aussage gilt für } n = 1 \\ \text{Schluß von } n - 1 \text{ auf } n \end{array} \right\} \text{ Aussage gilt für alle } n \in \mathbb{N}$

Türme von Hanoi

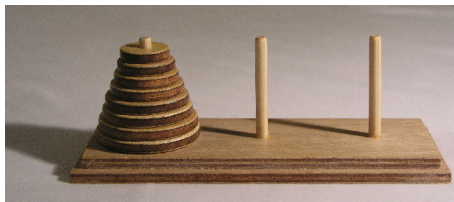


4.2 Rekursion

Vollständige Induktion: $\left. \begin{array}{l} \text{Aussage gilt für } n = 1 \\ \text{Schluß von } n - 1 \text{ auf } n \end{array} \right\} \text{ Aussage gilt für alle } n \in \mathbb{N}$

Türme von Hanoi

- 64 Scheiben, 3 Plätze, immer 1 Scheibe verschieben
- Ziel: Turm verschieben
- Es dürfen nur kleinere Scheiben auf größeren liegen.

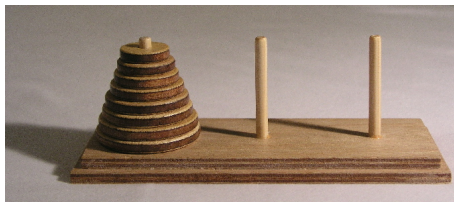


4.2 Rekursion

Vollständige Induktion: $\left. \begin{array}{l} \text{Aussage gilt für } n = 1 \\ \text{Schluß von } n - 1 \text{ auf } n \end{array} \right\} \text{ Aussage gilt für alle } n \in \mathbb{N}$

Türme von Hanoi

- 64 Scheiben, 3 Plätze, immer 1 Scheibe verschieben
- Ziel: Turm verschieben
- Es dürfen nur kleinere Scheiben auf größeren liegen.
- $n = 1$ Scheibe: fertig
- Wenn $n - 1$ Scheiben verschiebbar: schiebe $n - 1$ Scheiben auf Hilfsplatz, verschiebe die darunterliegende, hole $n - 1$ Scheiben von Hilfsplatz



4.2 Rekursion

Vollständige Induktion: $\left. \begin{array}{l} \text{Aussage gilt für } n = 1 \\ \text{Schluß von } n - 1 \text{ auf } n \end{array} \right\} \text{ Aussage gilt für alle } n \in \mathbb{N}$

Türme von Hanoi

- 64 Scheiben, 3 Plätze, immer 1 Scheibe verschieben
- Ziel: Turm verschieben
- Es dürfen nur kleinere Scheiben auf größeren liegen.
- $n = 1$ Scheibe: fertig
- Wenn $n - 1$ Scheiben verschiebbar: schiebe $n - 1$ Scheiben auf Hilfsplatz, verschiebe die darunterliegende, hole $n - 1$ Scheiben von Hilfsplatz

```
void verschiebe (int n, int start, int ziel)
{
    if (n == 1)
        verschiebe_1_scheibe (start, ziel);
    else
    {
        verschiebe (1, start, hilfsplatz);
        verschiebe (n - 1, start, ziel);
        verschiebe (1, hilfsplatz, ziel);
    }
}
```


4.2 Rekursion

Floodfill

```
void fill (int x, int y, char c, char o)
{
    if (get_point (x, y) == o)
    {
        put_point (x, y, c);
        fill (x + 1, y, c, o);
        fill (x - 1, y, c, o);
        fill (x, y + 1, c, o);
        fill (x, y - 1, c, o);
    }
}
```

4.2 Rekursion

Floodfill

```
void fill (int x, int y, char c, char o)
{
    if (get_point (x, y) == o)
    {
        put_point (x, y, c);
        fill (x + 1, y, c, o);
        fill (x - 1, y, c, o);
        fill (x, y + 1, c, o);
        fill (x, y - 1, c, o);
    }
}
```

Aufgabe: Schreiben Sie eine Bibliothek für „Text-Grafik“ mit folgenden Funktionen:

- **void clear (char c)**
Bildschirm auf Zeichen **c** löschen
- **void put_point (int x, int y, char c)**
Punkt setzen
- **char get_point (int x, int y)**
Punkt lesen
- **void fill (int x, int y, char c, char o)**
Fläche in der „Farbe“ **o**,
die den Punkt **(x, y)** enthält,
mit der „Farbe“ **c** ausmalen
- **void display (void)**
Inhalt des Arrays auf dem
Bildschirm ausgeben

Hinweis: Verwenden Sie ein Array als „Bildschirm“.

4.2 Rekursion

Ausgabe einer Hex-Zahl

- Den Rest der Zahl bei Division durch 16 ausgeben, ...
- ... *aber vorher* die Hex-Ziffern der durch 16 dividierten Zahl ausgeben.

4.2 Rekursion

Ausgabe einer Hex-Zahl

- Den Rest der Zahl bei Division durch 16 ausgeben, ...
- ... *aber vorher* die Hex-Ziffern der durch 16 dividierten Zahl ausgeben.

→ Array als Zwischenspeicher zum Umdrehen entfällt.

4.2 Rekursion

Ausgabe einer Hex-Zahl

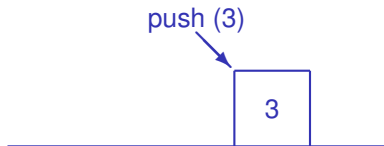
- Den Rest der Zahl bei Division durch 16 ausgeben, ...
- ... *aber vorher* die Hex-Ziffern der durch 16 dividierten Zahl ausgeben.

→ Array als Zwischenspeicher zum Umdrehen entfällt.

→ *Wirklich?*

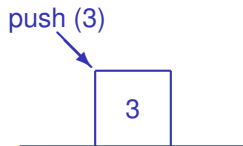
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

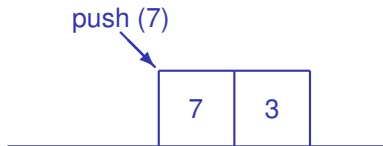
„Last In – First Out“



LIFO = Stack = Stapel

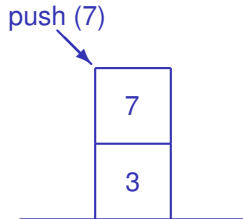
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“



LIFO = Stack = Stapel

4.3 Stack und FIFO

„First In – First Out“

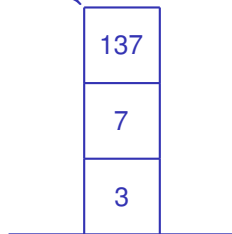
push (137)



FIFO = Queue = Reihe

„Last In – First Out“

push (137)



LIFO = Stack = Stapel

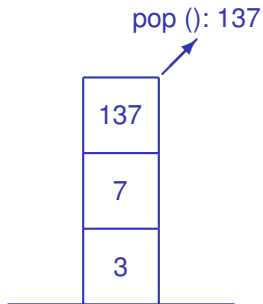
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

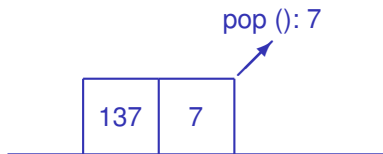
„Last In – First Out“



LIFO = Stack = Stapel

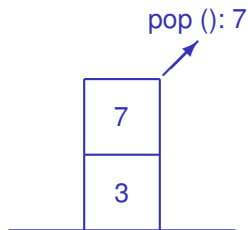
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

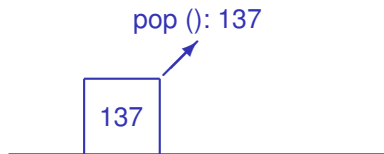
„Last In – First Out“



LIFO = Stack = Stapel

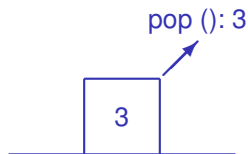
4.3 Stack und FIFO

„First In – First Out“



FIFO = Queue = Reihe

„Last In – First Out“



LIFO = Stack = Stapel

Angewandte Informatik

1 Einführung

2 Einführung in C

3 Bibliotheken

4 Algorithmen

4.1 Differentialgleichungen

4.2 Rekursion

4.3 Stack und FIFO

4.4 Aufwandsabschätzungen

5 Hardwarenahe Programmierung

...