

Angewandte Informatik

Prof. Dr. rer. nat. Peter Gerwinski

5. November 2015

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

...

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

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

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

Funktion wird „anderswo“ definiert

- separater C-Quelltext: mit an `gcc` übergeben
- 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.c -lGL -lGLU -lglut -o cube
```

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)
```

```
{
```

```
    ...
```

```
}
```

```
...
```

gedrückte Taste

Mausposition

```
glutKeyboardFunc (key_handler);
```

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

3.4 Projekt organisieren: make

- Regeln
- Makros

3.4 Projekt organisieren: make

- Regeln

```
philosophy: philosophy.o answer.o  
gcc philosophy.o answer.o -o philosophy
```

```
answer.o: answer.c  
gcc -c answer.c -o answer.o
```

```
philosophy.o: philosophy.c answer.h  
gcc -c philosophy.c -o philosophy.o
```

- Makros

3.4 Projekt organisieren: make

- Regeln
- Makros

```
PHILOSOPHY_SOURCES = philosophy.c answer.h  
PHILOSOPHY_OBJECTS = philosophy.o answer.o  
ANSWER_SOURCES = answer.c
```

```
philosophy: $(PHILOSOPHY_OBJECTS)  
    gcc $(PHILOSOPHY_OBJECTS) -o philosophy
```

```
answer.o: $(ANSWER_SOURCES)  
    gcc -c answer.c -o answer.o
```

```
philosophy.o: $(PHILOSOPHY_SOURCES)  
    gcc -c philosophy.c -o philosophy.o
```

```
clean:  
    rm -f $(PHILOSOPHY_OBJECTS) philosophy
```

3.4 Projekt organisieren: make

- Regeln
- Makros

→ 3 Sprachen: C, Präprozessor, make

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: Basketball

4.2 Rekursion

Vollständige Induktion:

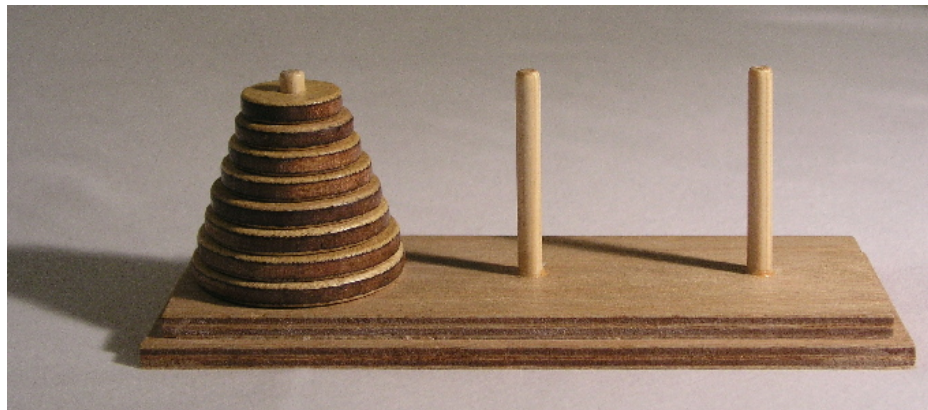
Aussage gilt für $n = 1$
Schluß von $n - 1$ auf n } Aussage gilt für alle $n \in \mathbb{N}$

4.2 Rekursion

Vollständige Induktion:

Aussage gilt für $n = 1$
Schluß von $n - 1$ auf n } Aussage gilt für alle $n \in \mathbb{N}$

Türme von Hanoi

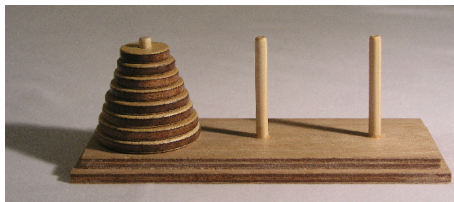


4.2 Rekursion

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.

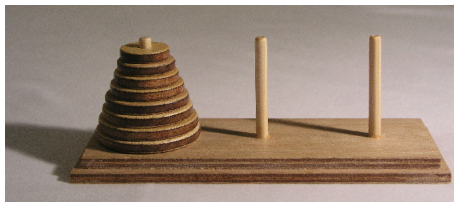


4.2 Rekursion

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“.

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

...