

Angewandte Informatik

Hardwarenahe Programmierung

Prof. Dr. rer. nat. Peter Gerwinski

21. November 2016

Angewandte Informatik

Hardwarenahe Programmierung

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Algorithmen
 - 4.1 Differentialgleichungen
 - 4.2 Rekursion
 - 4.3 Aufwandsabschätzungen
- 5 Hardwarenahe Programmierung
 - 5.1 Bit-Operationen
 - 5.2 I/O-Ports
 - 5.3 Interrupts
 - ...
- 6 Objektorientierte Programmierung

...

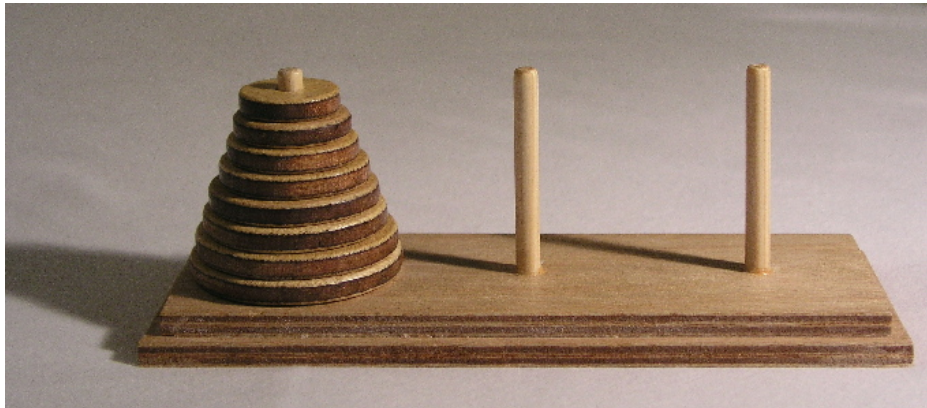
4.2 Rekursion

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

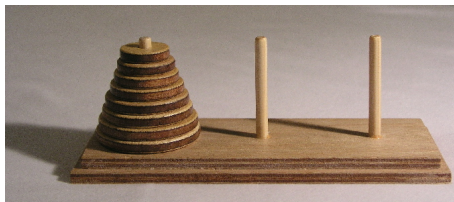


4.2 Rekursion

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.

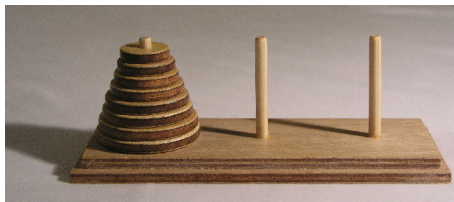


4.2 Rekursion

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 move (int from, int to, int disks)
{
    if (disks == 1)
        move_one_disk (from, to);
    else
    {
        int help = 0 + 1 + 2 - from - to;
        move (from, help, disks - 1);
        move (from, to, 1);
        move (help, to, disks - 1);
    }
}
```

4.3 **Aufwandsabschätzungen** – Komplexitätsanalyse

Beispiel: Sortieralgorithmen

- Minimum suchen

4.3 Aufwandsabschätzungen – Komplexitätsanalyse

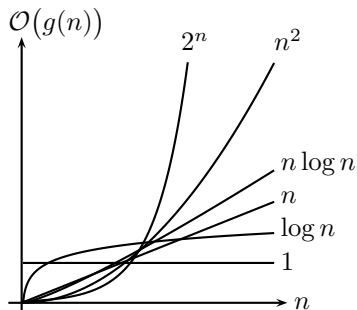
Beispiel: Sortieralgorithmen

- Minimum suchen
- Minimum an den Anfang tauschen,
nächstes Minimum suchen
→ Selectionsort

4.3 Aufwandsabschätzungen – Komplexitätsanalyse

Beispiel: Sortieralgorithmen

- Minimum suchen: $\mathcal{O}(n)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen
→ Selectionsort: $\mathcal{O}(n^2)$



n : Eingabedaten

$g(n)$: Rechenzeit

Faustregel:

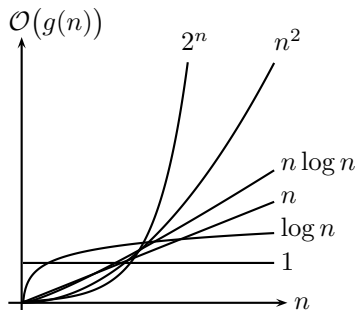
Schachtelung der Schleifen zählen

x Schleifen → $\mathcal{O}(n^x)$

4.3 Aufwandsabschätzungen – Komplexitätsanalyse

Beispiel: Sortieralgorithmen

- Minimum suchen: $\mathcal{O}(n)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen
→ Selectionsort: $\mathcal{O}(n^2)$
- Während Minimumsuche prüfen und abbrechen, falls schon sortiert
→ Bubblesort



n : Eingabedaten

$g(n)$: Rechenzeit

Faustregel:

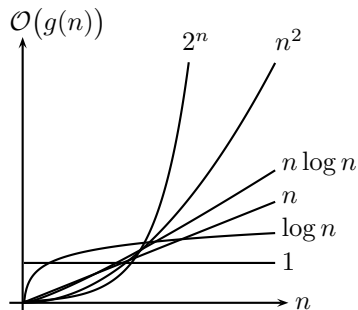
Schachtelung der Schleifen zählen

x Schleifen → $\mathcal{O}(n^x)$

4.3 Aufwandsabschätzungen – Komplexitätsanalyse

Beispiel: Sortieralgorithmen

- Minimum suchen: $\mathcal{O}(n)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen
→ Selectionsort: $\mathcal{O}(n^2)$
- Während Minimumsuche prüfen und abbrechen, falls schon sortiert
→ Bubblesort: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$



n : Eingabedaten

$g(n)$: Rechenzeit

Faustregel:

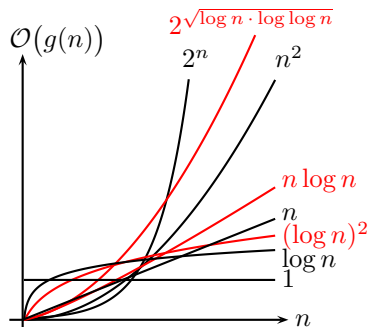
Schachtelung der Schleifen zählen

x Schleifen → $\mathcal{O}(n^x)$

4.3 Aufwandsabschätzungen – Komplexitätsanalyse

Beispiel: Sortieralgorithmen

- Minimum suchen: $\mathcal{O}(n)$
- Minimum an den Anfang tauschen, nächstes Minimum suchen
→ Selectionsort: $\mathcal{O}(n^2)$
- Während Minimumsuche prüfen und abbrechen, falls schon sortiert
→ Bubblesort: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$



n : Eingabedaten

$g(n)$: Rechenzeit

Faustregel:

Schachtelung der Schleifen zählen

x Schleifen $\rightarrow \mathcal{O}(n^x)$

RSA: Schlüsselerzeugung (Berechnung von d): $\mathcal{O}((\log n)^2)$,

Ver- und Entschlüsselung (Exponentiation): $\mathcal{O}(n \log n)$,

Verschlüsselung brechen (Primfaktorzerlegung): $\mathcal{O}(2^{\sqrt{\log n \cdot \log \log n}})$

Angewandte Informatik

Hardwarenahe Programmierung

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Algorithmen
 - 4.1 Differentialgleichungen
 - 4.2 Rekursion
 - 4.3 Aufwandsabschätzungen
- 5 Hardwarenahe Programmierung
 - 5.1 Bit-Operationen
 - 5.2 I/O-Ports
 - 5.3 Interrupts
 - ...
- 6 Objektorientierte Programmierung

...

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

5.1.1 Zahlensysteme

Basis	Name	Beispiel	Anwendung
2	Binärsystem	1 0000 0011	Bit-Operationen
8	Oktalsystem	0403	Dateizugriffsrechte (Unix)
10	Dezimalsystem	259	Alltag
16	Hexadezimalsystem	0x103	Bit-Operationen
256	(keiner gebräuchlich)	0.0.1.3	IP-Adressen (IPv4)

5.1.1 Zahlensysteme

Oktal- und Hexadezimal-Zahlen lassen sich ziffernweise in Binär-Zahlen umrechnen:

000	0	0000	0	1000	8
001	1	0001	1	1001	9
010	2	0010	2	1010	A
011	3	0011	3	1011	B
100	4	0100	4	1100	C
101	5	0101	5	1101	D
110	6	0110	6	1110	E
111	7	0111	7	1111	F

5.1.2 Bit-Operationen in C

C-Operator	Verknüpfung	Anwendung
&	Und	Bits gezielt löschen
	Oder	Bits gezielt setzen
^	Exklusiv-Oder	Bits gezielt invertieren
~	Nicht	Alle Bits invertieren
<<	Verschiebung nach links	Maske generieren
>>	Verschiebung nach rechts	Bits isolieren

Numerierung der Bits: von rechts ab 0

Bit Nr. 3 auf 1 setzen: `a |= 1 << 3;`

Bit Nr. 4 auf 0 setzen: `a &= ~(1 << 4);`

Bit Nr. 0 invertieren: `a ^= 1 << 0;`

Abfrage, ob Bit Nr. 1 gesetzt ist: `if (a & (1 << 1))`

5.1.2 Bit-Operationen in C

C-Datentypen für Bit-Operationen:

#include <stdint.h>

	8 Bit	16 Bit	32 Bit	64 Bit
mit Vorzeichen	int8_t	int16_t	int32_t	int64_t
ohne Vorzeichen	uint8_t	uint16_t	uint32_t	uint64_t

Ausgabe:

#include <stdio.h>

#include <stdint.h>

#include <inttypes.h>

...

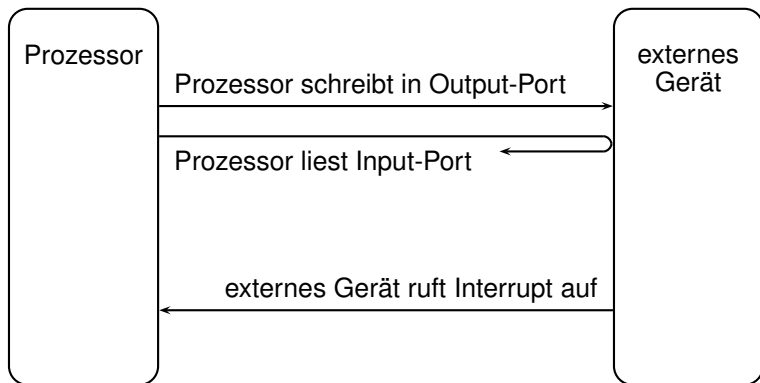
uint64_t x = 42;

printf ("Die_Antwort_lautet:_% " PRIu64 "\n", x);

5.2 I/O-Ports

5.3 Interrupts

Kommunikation mit externen Geräten



5.2 I/O-Ports

In Output-Port schreiben = Aktoren ansteuern

Beispiel: LED

```
#include <avr/io.h>
```

```
...
```

```
DDRC = 0x70;    binär: 0111 0000
```

```
PORTC = 0x40;   binär: 0100 0000
```

Herstellerspezifisch!

Details: siehe Datenblatt und Schaltplan

5.2 I/O-Ports

Aus Input-Port lesen = Sensoren abfragen

Beispiel: Taster

```
#include <avr/io.h>
```

```
...
```

```
DDRC = 0xfd;          binär: 1111 1101
```

```
while (PINC & 0x02 == 0) binär: 0000 0010
```

```
; /* just wait */
```

Herstellerspezifisch!

Details: siehe Datenblatt und Schaltplan

Angewandte Informatik

Hardwarenahe Programmierung

- 1 Einführung
- 2 Einführung in C
- 3 Bibliotheken
- 4 Algorithmen
 - 4.1 Differentialgleichungen
 - 4.2 Rekursion
 - 4.3 **Aufwandsabschätzungen**
- 5 **Hardwarenahe Programmierung**
 - 5.1 Bit-Operationen
 - 5.2 I/O-Ports
 - 5.3 **Interrupts**
 - ...
- 6 Objektorientierte Programmierung

...