

Angewandte Informatik

Prof. Dr. rer. nat. Peter Gerwinski

7. Januar 2016

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

4.4 Dynamische Speicherverwaltung

5 Hardwarenahe Programmierung

...

5.4 volatile-Variable

5.5 Software-Interrupts

5.6 Byte-Reihenfolge – Endianness

5.6 Speicherausrichtung – Alignment

6 Objektorientierte Programmierung

7 Ergänzungen und Ausblicke

5 Hardwarenahe Programmierung

5.1 Bit-Operationen

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

Auswendig lernen!

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

Beispiele:

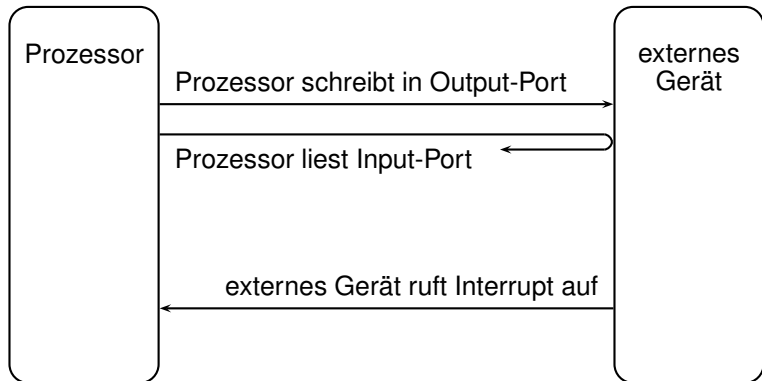
- Bit Nr. 5 gezielt auf 1 setzen: `PORTB |= 1 << 5;`
- Bit Nr. 6 gezielt auf 0 setzen: `PORTA &= ~(1 << 6);`
- Ist Bit Nr. 4 gesetzt? `if (PINC & (1 << 4) ...`
- Umschalten zwischen Ein- und Ausgabe: `DDR`
Bit = 1: Ausgabe; Bit = 0: Eingabe

**Details abhängig von
Prozessor und Compiler!**

5.2 I/O-Ports

5.3 Interrupts

Kommunikation mit externen Geräten



5.4 volatile-Variable

```
volatile uint16_t mleft_counter;
```

```
/* ... */
```

„Immer lesen und schreiben. Nicht wegoptimieren.“

```
volatile uint16_t mleft_dist;
```

```
/* ... */
```

```
ISR (INT0_vect)
```

```
{  
    mleft_dist++;  
    mleft_counter++;  
    /* ... */  
}
```

```
int main (void)
```

```
{  
    int prev_mleft_dist = mleft_dist;  
    while (mleft_dist == prev_mleft_dist)  
        /* just wait */;  
}
```

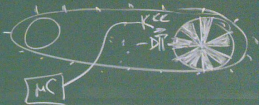
5.4 volatile-Variable

PORTA, PORTB, DDRA usw. sind **volatile**-Variable an numerisch vorgegebenen Speicheradressen (z. B. 0x38 für PORTB).

$$(*(\text{volatile uint8_t } *) (0 \times 18 + 0 \times 20))$$

(explizite Typumwandlg.) 0x38
in einen Zeiger
auf 8-Bit-Zahlen ohne Vorzeichen
ohne Zugriffsoptimierung

⇒ PORTB ≡ Speicherzelle Nr. 38₁₆



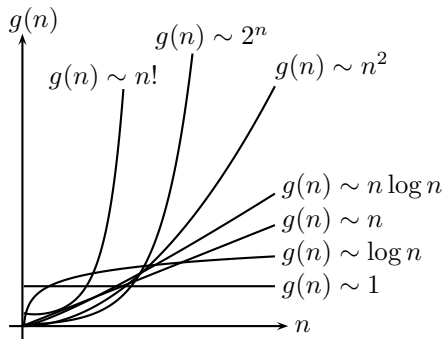
Zeiger dereferenzieren: Speicherzelle Nr. x direkt beschreiben →

Unwandlung einer Zahl in einen Zeiger
= Speicherzelle Nr. x
direkt ansprechen
hier: x = 0x38

5.5 Aufwandsabschätzungen

Beispiel: Sortieralgorithmen

- Maximum suchen: $\mathcal{O}(n)$



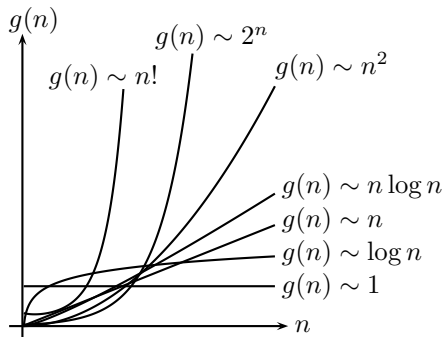
n : Eingabedaten

$g(n)$: Rechenzeit

5.5 Aufwandsabschätzungen

Beispiel: Sortieralgorithmen

- Maximum suchen: $\mathcal{O}(n)$
- Maximum ans Ende tauschen
→ Selectionsort: $\mathcal{O}(n^2)$



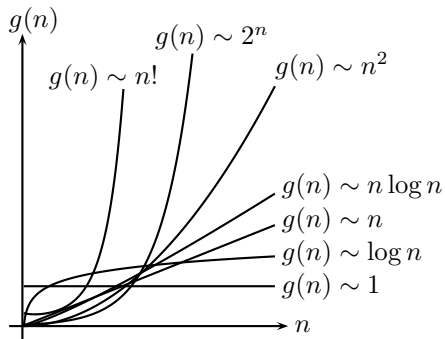
n : Eingabedaten

$g(n)$: Rechenzeit

5.5 Aufwandsabschätzungen

Beispiel: Sortieralgorithmen

- Maximum suchen: $\mathcal{O}(n)$
- Maximum ans Ende tauschen
→ Selectionsort: $\mathcal{O}(n^2)$
- zufällig mischen, bis sortiert
→ Monkeysort: $\mathcal{O}(n!)$



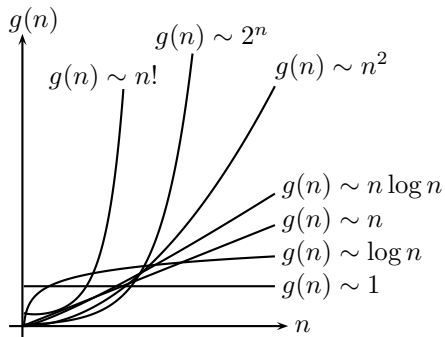
n : Eingabedaten

$g(n)$: Rechenzeit

5.5 Aufwandsabschätzungen

Beispiel: Sortieralgorithmen

- Maximum suchen: $\mathcal{O}(n)$
- Maximum ans Ende tauschen
→ Selectionsort: $\mathcal{O}(n^2)$
- zufällig mischen, bis sortiert
→ Monkeysort: $\mathcal{O}(n!)$
- Während Maximumsuche prüfen, abbrechen, falls schon sortiert
→ Bubblesort: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$



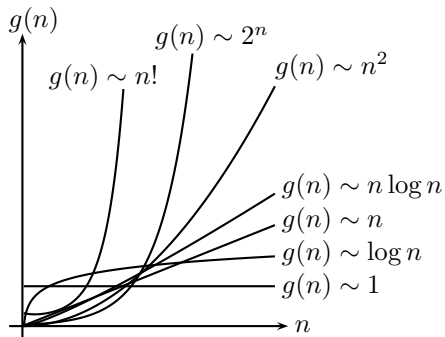
n : Eingabedaten

$g(n)$: Rechenzeit

5.5 Aufwandsabschätzungen

Beispiel: Sortialgorithmen

- Maximum suchen: $\mathcal{O}(n)$
- Maximum ans Ende tauschen
→ Selectionsort: $\mathcal{O}(n^2)$
- zufällig mischen, bis sortiert
→ Monkeysort: $\mathcal{O}(n!)$
- Während Maximumsuche prüfen, abbrechen, falls schon sortiert
→ Bubblesort: $\mathcal{O}(n)$ bis $\mathcal{O}(n^2)$
- Rekursiv sortieren
→ Quicksort: $\mathcal{O}(n \log n)$ bis $\mathcal{O}(n^2)$



n : Eingabedaten

$g(n)$: Rechenzeit

5.6 Dynamische Speicherverwaltung

```
char *name[] = { "Anna", "Berthold", "Caesar" };
```

```
...
```

```
name[3] = "Dieter";
```

5.6 Dynamische Speicherverwaltung

```
#include <stdlib.h>
```

```
...
```

```
char **name = malloc (3 * sizeof (char *));  
    /* Speicherplatz für 3 Zeiger anfordern */
```

```
...
```

```
free (name)  
    /* Speicherplatz freigeben */
```

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

4.4 Dynamische Speicherverwaltung

5 Hardwarenahe Programmierung

...

5.4 volatile-Variable

5.5 Software-Interrupts

5.6 Byte-Reihenfolge – Endianness

5.6 Speicherausrichtung – Alignment

6 Objektorientierte Programmierung

7 Ergänzungen und Ausblicke