

Angewandte Informatik

Prof. Dr. rer. nat. Peter Gerwinski

21. Januar 2016

Angewandte Informatik

- 1 Einführung**
- 2 Einführung in C**
- 3 Bibliotheken**
- 4 Algorithmen**
- 5 Hardwarenahe Programmierung**
 - ...
 - 5.4** volatile-Variable
 - 5.5** Software-Interrupts
 - 5.6** Byte-Reihenfolge – Endianness
 - 5.6** Speicherausrichtung – Alignment
- 6 Objektorientierte Programmierung**
 - 6.1** Konzepte und Ziele
 - 6.2** Beispiel: Zahlen und Buchstaben
 - 6.3** Einführung in C++
- 7 Ergänzungen und Ausblicke**

5.5 Software-Interrupts

`mov ax, 0012` ← Parameter in Prozessorregister
`int 10` ← Funktionsaufruf über Interrupt-Vektor

Beispiel: VGA-Grafikkarte

- Modus setzen: `mov ah, 00`
- Grafikmodus: `mov al, 12`
- Textmodus: `mov al, 03`

Verschiedene Farben: Output-Ports

- *Graphics Register*: Index `03CE`, Daten `03CF`
- Index 0: *Set/Reset Register*
- Index 1: *Enable Set/Reset Register*
- Index 8: *Bit Mask Register*
- Jedes Bit steht für Schreibzugriff auf eine Speicherbank.
- 4 Speicherbänke → 16 Farben

5.6 Byte-Reihenfolge – Endianness

5.6.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.

Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

$$1027 = 1024 + 2 + 1 = 0000\ 0100\ 0000\ 0011_2 = 0403_{16}$$

Speicherzellen:

04	03
----	----

 Big-Endian „großes Ende zuerst“
für Menschen leichter lesbar

03	04
----	----

 Little-Endian „kleines Ende zuerst“
bei Additionen effizienter

→ Geschmackssache

... **außer bei Datenaustausch!**

5.6 Byte-Reihenfolge – Endianness

5.6.1 Konzept

Eine Zahl geht über mehrere Speicherzellen.
Beispiel: 16-Bit-Zahl in 2 8-Bit-Speicherzellen

Welche Bits liegen wo?

—→ Geschmackssache

... **außer bei Datenaustausch!**

- Dateiformate
- Datenübertragung

5.6 Byte-Reihenfolge – Endianness

5.6.2 Dateiformate

Audio-Formate: Reihenfolge der Bytes in 16- und 32-Bit-Zahlen

- RIFF-WAVE-Dateien (.wav): Little-Endian
- Au-Dateien (.au): Big-Endian
- ältere AIFF-Dateien (.aiff): Big-Endian
- neuere AIFF-Dateien (.aiff): Little-Endian

Grafik-Formate: Reihenfolge der Bits in den Bytes

- PBM-Dateien: Big-Endian, MSB first
- XBM-Dateien: Little-Endian, LSB first

MSB/LSB = most/least significant bit

5.6 Byte-Reihenfolge – Endianness

5.6.3 Datenübertragung

- RS-232 (serielle Schnittstelle): LSB first
- I²C: MSB first
- USB: beides
- Ethernet: LSB first
- TCP/IP (Internet): Big-Endian

5.7 Speicherausrichtung – Alignment

```
#include <stdint.h>
```

```
uint8_t a;  
uint16_t b;  
uint8_t c;
```

Speicheradresse durch 2 teilbar – „16-Bit-Alignment“

- 2-Byte-Operation: effizienter
- ... oder sogar nur dann erlaubt

→ Compiler optimiert Speicherausrichtung

```
uint8_t a;  
uint8_t dummy;  
uint16_t b;  
uint8_t c;  
  
uint8_t a;  
uint8_t c;  
uint16_t b;
```

Fazit:

- Adressen von Variablen sind systemabhängig
- Bei Definition von Datenformaten Alignment beachten → effizienter

6 Objektorientierte Programmierung

6.1 Konzepte und Ziele

- Array: feste Anzahl von Elementen desselben Typs (z. B.: 3 Zeiger)
- Dynamisches Array: variable Anzahl von Elementen desselben Typs
- Problem: Elemente unterschiedlichen Typs
- Lösung: den Typ des Elements zusätzlich speichern

6 Objektorientierte Programmierung

6.1 Konzepte und Ziele

- Problem: Elemente unterschiedlichen Typs
- Lösung: den Typ des Elements zusätzlich speichern
- Zeiger auf verschiedene Strukturen mit einem gemeinsamen Anteil von Datenfeldern
→ „verwandte“ *Objekte*, *Klassen* von Objekten
- Struktur, die *nur* den gemeinsamen Anteil enthält
→ „Vorfahr“, *Basisklasse*, *Vererbung*
- Explizite Typumwandlung eines Zeigers auf die Basisklasse in einen Zeiger auf die *abgeleitete Klasse*
→ Man kann ein Array unterschiedlicher Objekte in einer Schleife abarbeiten.
→ *Polymorphie*

6 Objektorientierte Programmierung

6.1 Konzepte und Ziele

- *Objekte, Klassen, Basisklassen, abgeleitete Klassen*
- *Vererbung, Polymorphie*
- Funktionen, die mit dem Objekt arbeiten: *Methoden*
- Aufgerufene Funktion hängt vom Typ des Objekts ab: *virtuelle Methode*
- Realisierung über Zeiger, die im Objekt gespeichert sind
(Genaugenommen: Tabelle von Zeigern)

6 Objektorientierte Programmierung

6.2 Beispiel: Zahlen und Buchstaben

```
typedef struct
{
    int type;
} t_base;
```

```
typedef struct
{
    int type;
    int content;
} t_integer;
```

```
typedef struct
{
    int type;
    char *content;
} t_string;
```

```
typedef struct
{
    int type;
} t_base;
```

```
typedef struct
{
    int type;
    int content;
} t_integer;
```

```
typedef struct
{
    int type;
    char *content;
} t_string;
```

```
t_integer i = { 1, 42 };
t_string s = { 2, "Hello,_world!" };
```

```
t_base *object[] = { (t_base *) &i, (t_base *) &s };
```


explizite

Typumwandlung

```
typedef struct  
{  
    int type;  
} t_base;
```

```
typedef struct  
{  
    int type;  
    int content;  
} t_integer;
```

```
typedef struct  
{  
    int type;  
    char *content;  
} t_string;
```

```
typedef union  
{  
    t_base base;  
    t_integer integer;  
    t_string string;  
} t_object;
```

typedef struct

```
{  
    void (* print) (union t_object *this);  
} t_base;
```

typedef struct

```
{  
    void (* print) (...);  
    int content;  
} t_integer;
```

typedef struct

```
{  
    void (* print) (union t_object *this);  
    char *content;  
} t_string;
```

typedef union

```
{  
    t_base base;  
    t_integer integer;  
    t_string string;  
} t_object;
```

object[i]—>base.print (object[i]);

6 Objektorientierte Programmierung

6.2 Beispiel: Zahlen und Buchstaben

Weitere Beispiele:

- Editor für graphische Objekte
- Datenbank-Software
- graphische Benutzeroberfläche (GUI)

6 Objektorientierte Programmierung

6.3 Einführung in C++

```
typedef struct  
{  
    void (* print) (union t_object *this);  
} t_base;
```

```
typedef struct  
{  
    void (* print) (...);  
    int content;  
} t_integer;
```

```
typedef struct  
{  
    void (* print) (union t_object *this);  
    char *content;  
} t_string;
```

6 Objektorientierte Programmierung

6.3 Einführung in C++

```
struct TBase  
{  
    virtual void print (void);  
};
```

```
struct TInteger: public TBase  
{  
    virtual void print (void);  
    int content;  
};
```

```
struct TString: public TBase  
{  
    virtual void print (void);  
    char *content;  
};
```

7 Ergänzungen und Ausblicke

7.1 String-Operationen

```
#include <string.h>
```

```
if (strcmp (s1, s2) < 0)  
    printf ("%s_ist_alphabetisch_kleiner_als_%s.\n", s1, s2);
```

```
strlen ()
```

```
strcpy ()
```

```
strncpy ()
```

```
...
```

7 Ergänzungen und Ausblicke

7.2 Dateien

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

```
FILE *f = fopen ("fhello.txt", "w");  
fprintf (f, "Hello, world!\n");  
fclose (f);
```

7 Ergänzungen und Ausblicke

7.3 Wie geht es weiter?

- das Gelernte anwenden
- weitere Algorithmen und Bibliotheken
- Vertiefung C++
- ...

Empfehlung:

- lesen, was andere geschrieben haben
- Fehler verbessern
- eigene Vorstellungen realisieren

Angewandte Informatik

- 1 Einführung**
- 2 Einführung in C**
- 3 Bibliotheken**
- 4 Algorithmen**
- 5 Hardwarenahe Programmierung**
 - ...
 - 5.4** volatile-Variable
 - 5.5** Software-Interrupts
 - 5.6** Byte-Reihenfolge – Endianness
 - 5.6** Speicherausrichtung – Alignment
- 6 Objektorientierte Programmierung**
 - 6.1** Konzepte und Ziele
 - 6.2** Beispiel: Zahlen und Buchstaben
 - 6.3** Einführung in C++
- 7 Ergänzungen und Ausblicke**

Angewandte Informatik

- 1 Einführung**
- 2 Einführung in C**
- 3 Bibliotheken**
- 4 Algorithmen**
- 5 Hardwarenahe Programmierung**
 - ...
 - 5.4** volatile-Variable
 - 5.5** Software-Interrupts
 - 5.6** Byte-Reihenfolge – Endianness
 - 5.6** Speicherausrichtung – Alignment
- 6 Objektorientierte Programmierung**
 - 6.1** Konzepte und Ziele
 - 6.2** Beispiel: Zahlen und Buchstaben
 - 6.3** Einführung in C++
- 7 Ergänzungen und Ausblicke**