Haute école d'ingénierie et d'architecture Fribourg
Hochschule für Technik und Architektur Freiburg

HTA-FR – Kurs Telekommunikation

Verfasser:
D. Gachet / HTA-FR - Telekommunikation

Embedded systems 1 und 2

Programmiersprache C – Zeiger

Klasse T-2 // 2018-2019

Inhalt

- Deklaration
- Zuordnungen und Dereferenzierungen
- Operatoren und Komparatoren
- Zugriff auf Peripheriegeräte
- Dynamische Objekte
- Typenumwandlung

Deklaration

Zeiger sind für eine höhere Programmiersprache sehr wichtige Elemente. Der Zeiger ist eine Variable, die die Adresse enthält, an der ein anderes Datenelement gespeichert ist.

Zeiger werden in C sehr oft verwendet, weil sie Folgendes ermöglichen:

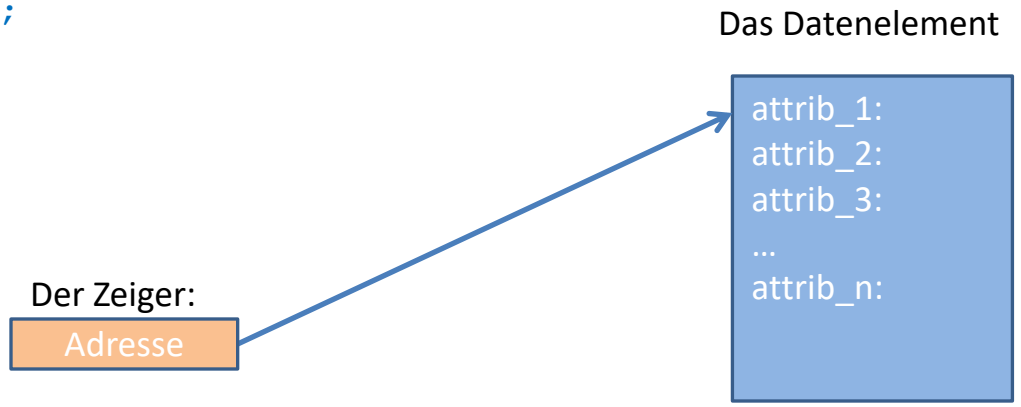
- Die Rückgabe von Ergebnissen bei der Verarbeitung von Information durch eine Funktion
- Das dynamische Erzeugen und Verarbeiten von Objekten
- Den Zugriff auf die Register von Peripheriegeräten (Hardware-Register)

Die Deklaration eines Zeigers hat die folgende Form:

```
<type_name>* <variable_name>;
```

Der `'*'` zeigt an, dass die Variable ein Zeiger auf das Datenelement vom Typ `<type_name>` ist, z. B.:

```
struct point* ppoint;
```



Die Adresse eines Objekts kann leicht mit dem Operator '&' erhalten und einer Variablen vom Typ Zeiger zugeteilt werden, z. B.:

```
char    avalue = 'a';
char    *pvalue;
```

```
section   .data
avalue:   .byte 'a'

section   .bss
pvalue:   .space 4
```

avalue: 'a' 0xa400001c

pvalue: 0x00 | 0x00 | 0x00 | 0x00 0xa0000020

```
pvalue = &avalue;
```

```
ldr     r0,=avalue
ldr     r1,=pvalue
str     r0,[r1]
```

avalue: 'a' 0xa400001c

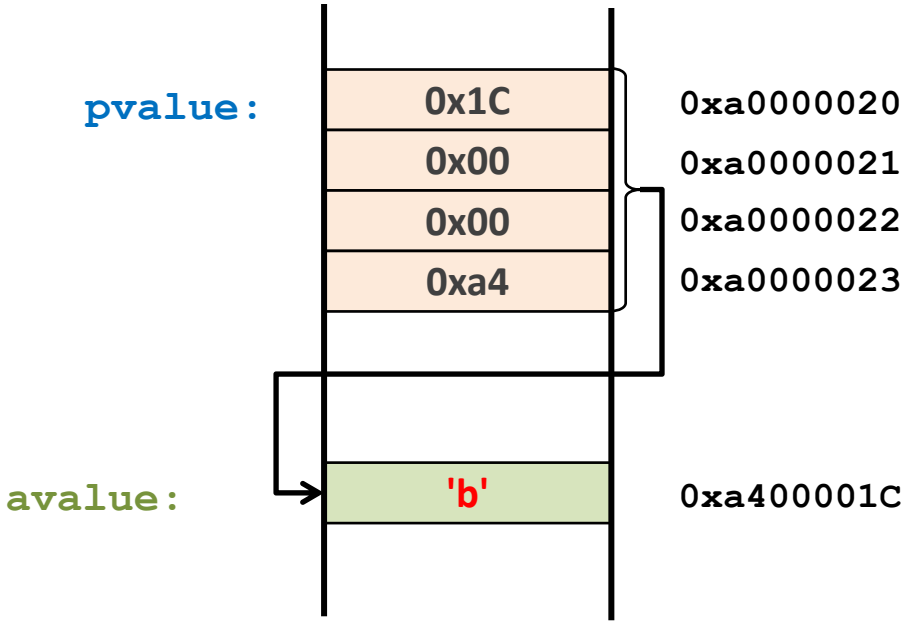
pvalue: 0xa4 | 0x00 | 0x00 | 0x1c 0xa0000020

Dereferenzierung: Basistypen

Der Operator '*' erlaubt den Zugriff auf den Inhalt, der durch den Zeiger referenziert wird
(indirekter Zugriff – Dereferenzierung), z. B.:

```
*pvalue = 'b';
```

```
ldr     r1,=pvalue
ldr     r5,[r1]
mov     r6,#'b'
strb    r6,[r5]
```



Wenn ein Zeiger ein Datenelement vom Typ Strukturur '`struct`' referenziert, kann der Zugriff auf die Komponenten/Attribute wie folgt realisiert werden:

```
struct point apoint;
struct point* ppoint = &apoint
```

```
(*ppoint).x = 10;
ppoint->y = 30;
```

äquivalent

```
apoint.x = 10;
apoint.y = 30;
```

Anmerkung:

Die Form `int x = *ppoint.x` ist ungültig. Tatsächlich interpretiert der Compiler die Zuordnung als `int x = *(ppoint.x)`, indem er die Komponente `x` und nicht `ppoint` dereferenziert.

Dereferenzierung: Tabellen und Zeiger

C macht nur kleine Unterschiede zwischen den Zeigern und den Tabellen. Sie erlaubt auf diese Weise den Zugriff auf die Elemente einer Tabelle, indem ein Zeiger dereferenziert wird und umgekehrt, z. B.:

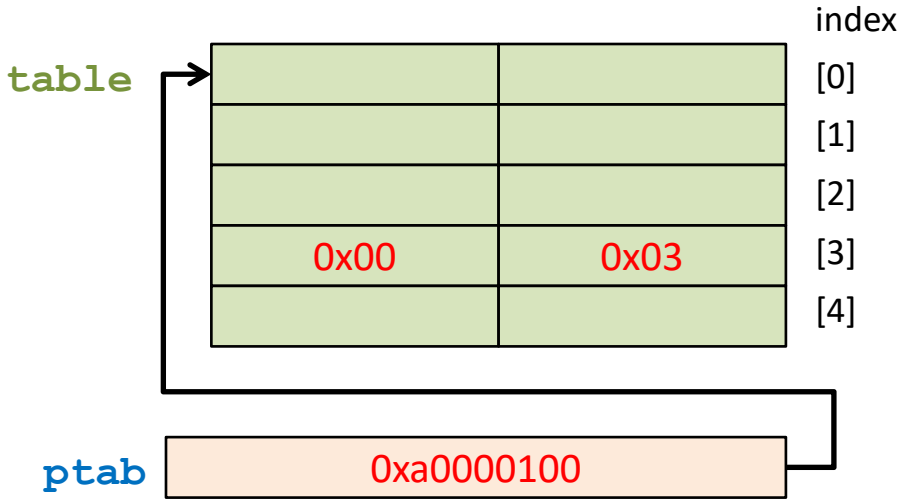
```
short  table[5];
short *ptab;
```

```
section .bss
table:  .space 5*2
ptab:   .space 4
```

```
ptab = table;
ptab[3] = 3;
```

```
ldr     r0,=table
ldr     r1,=ptab
str     r0,[r1]

ldr     r3,[r1]
mov     r4,#3
strh    r4,[r3,#6]
```



Dereferenzierung: Tabellen und Zeiger (II)

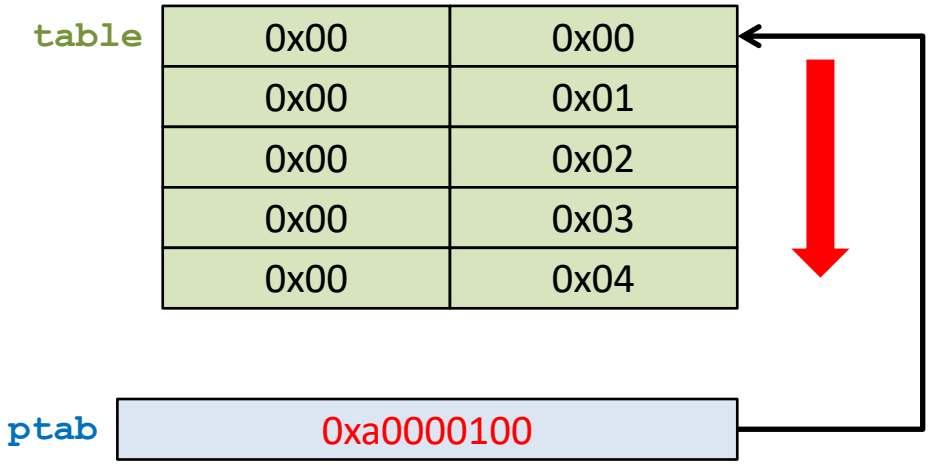
```
ptab = table;
for(int i=0; i<5; i++)
{
    *ptab++ = i;
}
```

```
        ldr     r0,=table
        ldr     r1,=ptab
        str     r0,[r1]

        mov     r2, #0
        ldr     r3,[r1]
        b       test

loop:   strh    r2,[r3],#2
        add     r2,#1

test:   cmp     r2,#5
        blo     loop
```



Arithmetische Operatoren und Komparatoren

Operatoren:

`+`	Addition
`-`	Subtraktion
`++`	Inkrementierung
`--`	Dekrementierung

Komparatoren:

`==`	gleich
`!=`	ungleich
`<`	kleiner als
`>`	grösser als
`<=`	kleiner oder gleich
`>=`	grösser oder gleich

Anmerkung

Die Zeigerarithmetik berücksichtigt zur Ausführung der Operation die Grösse des referenzierten Datentyps, z. B.

```
char *pc = (char*)100;  pc++;  // → pc == 101
long *pl = (long*)100;  pl++;  // → pl == 104
```

Zugriff auf Peripherieeinheiten

Wenn ein Zeiger die Register eines Peripheriegeräts mit direktem Speicherzugriff (memory mapped device) referenziert, nimmt die Speicheradresse, an der sich die Register befinden, die folgende Form an:

```
struct hw_regs {
  uint16_t reg1;
  uint16_t reg2;
  uint16_t reg3;
  uint16_t reg4;
};
volatile struct hw_regs * hwreg = (struct hw_regs *)0xd0000000;
```

Dynamische Objekte

Die Zeiger erlauben die dynamische Manipulation der Objekte. Diese lassen sich einfach mithilfe der Methoden "`malloc`"/"`calloc`" und "`free`" der Standard Bibliothek `<stdlib.h>` erzeugen und löschen, z. B.:

```
// declaration of the operators malloc/calloc & free
#include <stdlib.h>

// creation of dynamic objects
struct point* ppoint = malloc (sizeof(struct point));
// oder
struct point* ppoint2 = calloc (1, sizeof(*ppoint2));

// deletion of dynamic objects
free (ppoint);
free (ppoint2);
```

Die Konstante `NULL` oder der Wert `0` zeigt an, dass der Zeiger kein Objekt referenziert (`NULL` ist definiert in `<stddef.h>`).

Typenumwandlung

C erlaubt die Umwandlung des Datenwerts eines Typs (Quelle) in einen andern Typ (Ziel). Man spricht auch von Zwang oder auf Englisch "type casting".

C unterscheidet zwei Umwandlungstypen, die implizite und die explizite Umwandlung.

Die implizite Umwandlung wird bei der Auswertung von Ausdrücken benutzt, die Basitypen der Sprache enthalten, z. B. werden die `char` und die `short` in `int` umgewandelt.

Die explizite Umwandlung wird durch die Programmierer oft bei der Manipulation von Zeigern und dynamischen Objekten verwendet. Die einfachste Form ist die Verwendung von `union`, die es erlaubt, eine Quelle unter verschiedenen Typen zu sehen. Die zweite hat die folgende Form:

```
type_a value_a;
type_b value_b = (type_b) value_a;
```