Prozedurale Programmierung

Fortgeschrittene Zeigerkonzepte

Hochschule Rosenheim - University of Applied Sciences

WS 2018/19

Prof. Dr. F.J. Schmitt

Überblick

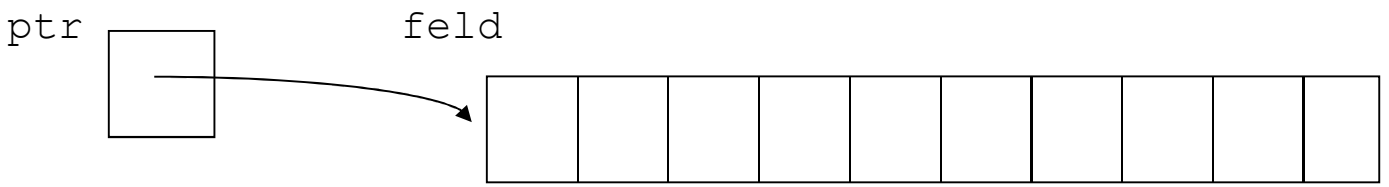
- Zeiger und Felder
- Zeigerarithmetik
- Zeiger auf Funktionen

Dualität von Zeigern und Feldern (1)

- Software-technisch gesehen sind Zeiger und Felder zwei völlig verschiedene Konstrukte
- In C kann jedoch der Name eines Feldes als Zeiger verwendet werden:

```
long feld[10];
long *ptr;

ptr = &feld[0];
```

- Zuweisung möglich, da &feld[0] die Adresse des ersten Elements des Feldes ist



Dualität von Zeigern und Feldern (2)

- Leider oft verwirrend
- Zeiger = Variable, die eine Adresse enthält
  - sizeof-Operator liefert die Anzahl der Bytes, die auf der jeweiligen Rechnerarchitektur zur Speicherung einer Adresse verwendet werden (32-Bit-Architektur = 4 Byte; 64-Bit Architektur = 8 Byte)
- Feld = Verbundtyp, d.h. eine Menge von Werten desselben Datentyps können gespeichert werden.
  - sizeof-Operator liefert den Speicherverbrauch des Feldes in Bytes

Dualität von Zeigern und Feldern (2)

- Austausch zweier Felder:
  - Geg:

```
long feld1[10];
long feld2[10];
```

  - Ges: Austausch aller Elemente von feld1 und feld2

  - Lösung:
    - Verwendung Feldern:
      elementweises Austauschen ist notwendig => zeitaufwändig
    - Verwendung von Zeigern:

```
ptr1 = &feld1[0];
ptr2 = &feld2[0];

ptr1 = &feld2[0];
ptr2 = &feld1[0];
```

Zeigerarithmetik (1)

- Ausgangssituation:

```
long feld[10];
long *ptr;
ptr = &feld[0];
```

- Adresse des n-ten Elements:

```
&feld[n]  äquivalent zu ptr + n;
```

- Wird ein Zeiger um `n` erhöht, so wird seine Adresse auf das `n`-te im Speicher unmittelbar folgende Objekt vom selben Typ gesetzt – Objektgröße wird berücksichtigt!

Zeigerarithmetik (2)

	Zeigerschreibweise	Feldschreibweise
Adresse	`ptr + n`	`&feld[n]`
Objekt	`*(ptr + n)`	`feld[n]`

- Ist `ptr` ein Zeiger auf `long`, so bewirkt der Ausdruck

```
ptr = ptr + 1;
```

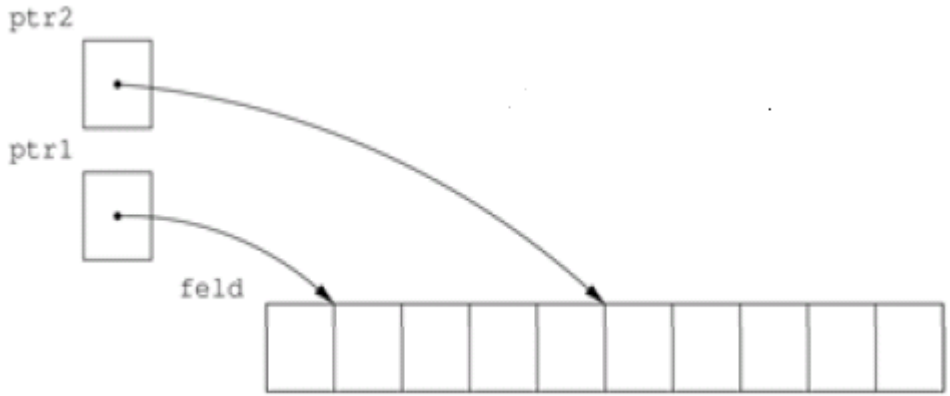
dass die Adresse in `ptr` um 4 Byte erhöht wird

Zeigerarithmetik (3)

```
long *ptr, *ap;
long v;
long feld[3];

v = 0;
ptr = &v;  // Zeige auf v
*ptr = 5;  // Überschreibe Objekt, auf das ptr zeigt mit 5

feld[0] = 2;
ptr = &feld[0]; // ptr zeigt auf Adresse von feld[0]
*ptr = 3;    // Überschreibe Objekt, auf das ptr zeigt mit 3

*ptr = *ptr + 1; // Erhöhe das Objekt, auf welches ptr zeigt,
                 // um 1: feld[0] = 4
ptr = ptr + 1;   // Erhöhe Adresse von ptr um 1 Element:
                 // ptr zeigt auf feld[1]
ap = ptr;        // ap zeigt auch auf feld[1]
```

Zeigerarithmetik (4)

- Differenz zweier Zeiger liefert die Anzahl der Objekte, die zwischen den Zeigern liegen:

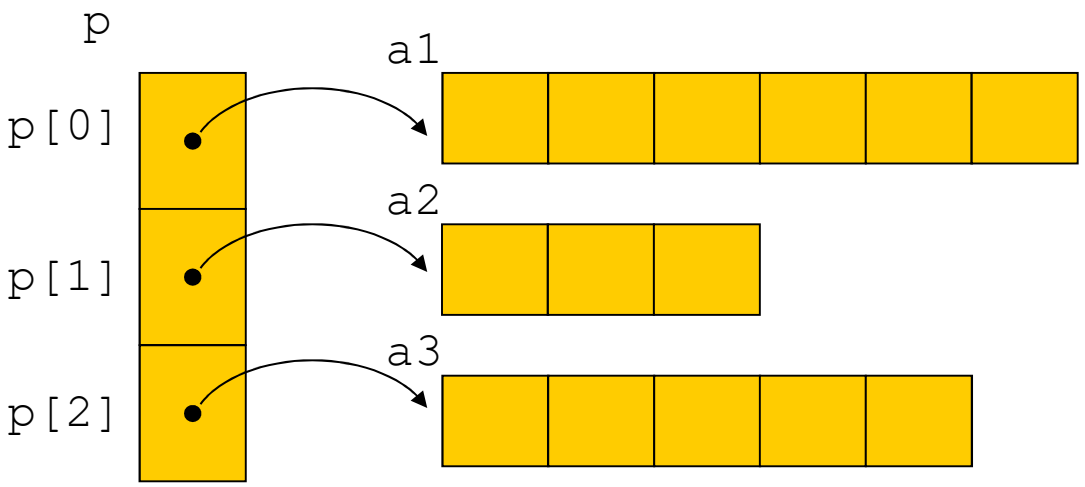


  - Zeiger ptr2 hat höhere Adresse als Zeiger ptr1
  - Differenz ergibt 4

Komplexere Fälle (1)

- Felder von Zeigern
  - Geeignet für die Verwaltung von vielen Feldern unterschiedlicher Länge

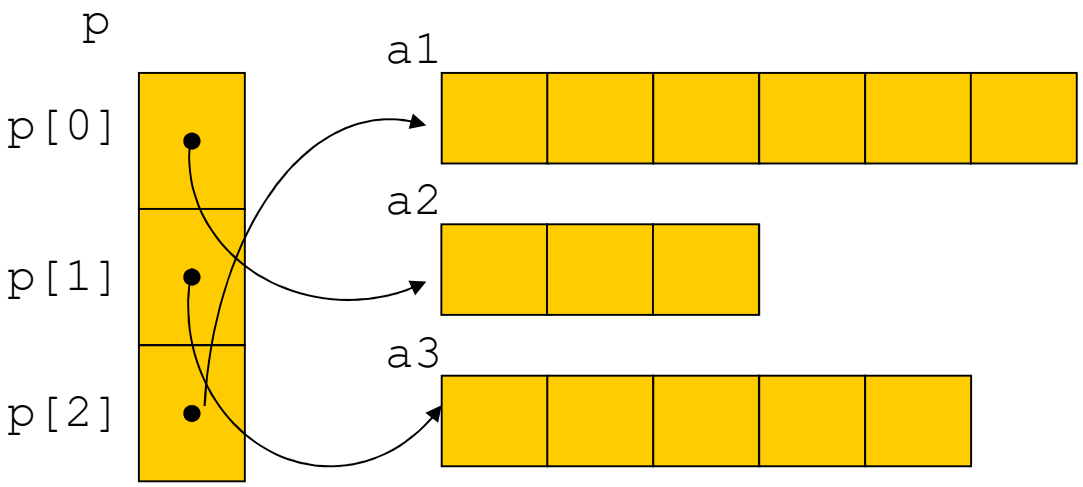
```
long a1[6];
long a2[3];
long a3[5];

long *p[3];

p[0] = &a1[0];
p[1] = &a2[0];
p[2] = &a3[0];
```



  - Zu jedem Feld muss auch seine Länge verwaltet werden!

Komplexere Fälle (2)

- Felder von Zeigern
  - Wie können die Positionen der Felder a1, a2 und a3 vertauscht werden?



  - Änderung der Reihenfolge durch Tauschen von Zeigern

Komplexere Fälle (3)

- Zeiger auf Zeiger
  - Zeiger können auf beliebigen Datentyp zeigen => auch auf Zeiger
  - Konstrukte wie „Zeiger auf Zeiger auf Felder von Zeigern“ können geschaffen werden
  - Bei unüberlegten Einsatz solcher Konstrukte kann ein Programm aber schnell unleserlich und nicht mehr wartbar sein

Komplexere Fälle (4)

- Zeiger auf Zeiger
  - Beispiel: Tauschen von zwei Zeigern

```
long v1;
long v2;

long *ptr1;
long *ptr2;

ptr1 = &v1;
ptr2 = &v2;
//...
SwapZeiger(&ptr1, &ptr2);
//...
```

Komplexere Fälle (5)

- Zeiger auf Zeiger
  - Funktion `SwapZeiger`

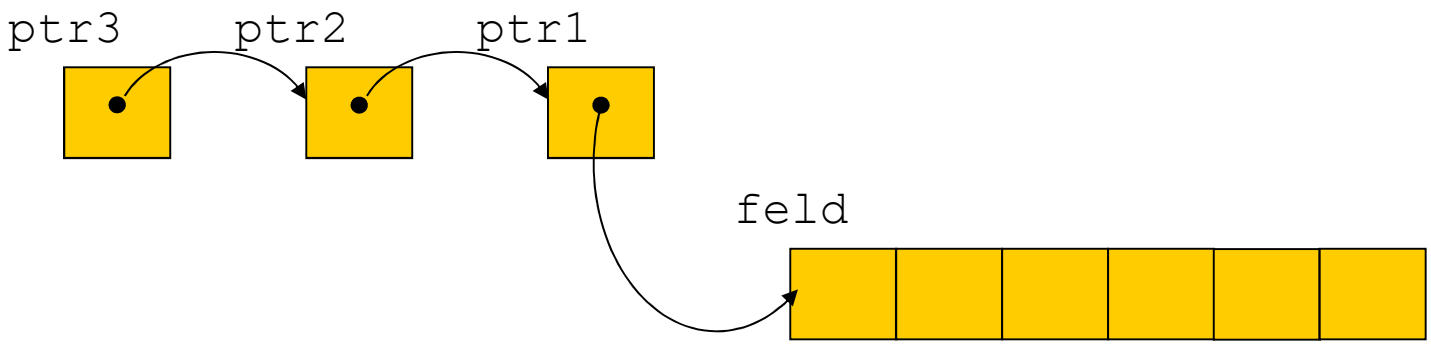
```
void SwapZeiger(long **p1, long **p2)
{
  long *h;

  //Dreieckstausch
  h = *p1;
  *p1 = *p2;
  *p2 = h;
}
```

  - Adressen der zwei zu tauschenden Variablen müssen angegeben werden (Zeiger auf die Objekte vom Typ `long *`)

Komplexere Fälle (6)

- Zeiger auf Zeiger
  - Nicht zu empfehlen:

```
long feld[6];
long *ptr1   = &feld[0];
long **ptr2  = &ptr1;
long ***ptr3 = &ptr2:
```



Zeiger auf Funktionen (1)

- In C können auch Zeiger auf Funktionen gesetzt werden
- Nicht nur Daten haben Adressen, sondern auch Funktionen

```
void HalloWelt()
{
  printf("Hallo Welt!");
}
```

- Adresse der Funktion:

```
&HalloWelt;
```

Anm.: auch der Funktionsname alleine steht für die Adresse der Funktion (nicht empfohlen)

Zeiger auf Funktionen (2)

- Vorsicht: Verwechslungsgefahr mit Funktionsaufruf

```
&HalloWelt; // Adresse der Funktion (empfohlen)

HalloWelt; // Adresse der Funktion (nicht empfohlen)

HalloWelt(); // Aufruf der Funktion Hallo Welt
```

Zeiger auf Funktionen (3)

- Definition

```
Typ (*Funktionsname) (Parameterliste);
```

  - Unterscheidet sich nur durch Klammernpaar und * von Definition einer Funktion

  - Beispiel:

```
long (*fptr) (long, long); // Zeiger auf eine
                              Funktion

long *funk (long, long);   // Deklaration einer
                              Funktion
```

Zeiger auf Funktionen (4)

- Verwendung
  - Können ähnlich wie „normale“ Zeiger gesetzt und kopiert werden
  - Bei Zuweisungen ist darauf zu achten, dass die Datentypen übereinstimmen
  - Beispiel:

```
long (*fptr) (long, long); // Zeiger auf eine
                           Funktion
```

Zeiger auf Funktionen (5)

- Verwendung
  - Ferner sind weitere zwei Funktionen gegeben:

```
long Summe (long x, long y)
{
  return x + y;
}

long Differenz (long x, long y)
{
  return x - y;
}
```

Zeiger auf Funktionen (5)

- Verwendung

```
...
long erg;

fptr = &Summe;
erg = fptr(1, 2);

fptr = &Differenz;
erg = fptr(1, 2);
```

  - Mit dem gleichen Aufruf werden zwei unterschiedliche Funktionen aufgerufen
  - Allein am Aufruf nicht erkennbar welche Funktion aufgerufen wird