Pointer und Pointer-Arithmetik

Praktikum „C-Programmierung“

Eugen Betke, Nathanael Hübbe,Michael Kuhn, Jakob Lüttgau, Jannek Squar

Wissenschaftliches Rechnen
Fachbereich Informatik
Universität Hamburg

2018-11-19

Universität Hamburg
DER FORSCHUNG | DER LEHRE | DER BILDUNG

Allgemein

- Ein Pointer
  - Zeigt auf eine Variable die in der Deklaration angegeben wurde
  - Pointer von verschiedenen Typen sind nicht gleich
  - Es gibt keine implizite Umwandung von einem Typ in einen anderen (wie bei arithmetischen Typen)
  - Ein nicht initializierter Pointer zeigt auf eine zufällige Adresse

```
1 int *ip; // ip ist vom Typ "pointer to int" und zeigt eine int Variable
```

Operatoren: address-of und object-of

- `&` address-of Operator
  - Liefert die Adresse einer Variable
  - Der Rückgabewert Pointer-Typ
- `*` object-of Operator
  - Ermöglicht den Zugriff auf ein Objekt durch seine Adresse

Beispiel[1]

```
1  int ar[20], *ip;
```

- Array und Pointer haben noch keine Verbindung

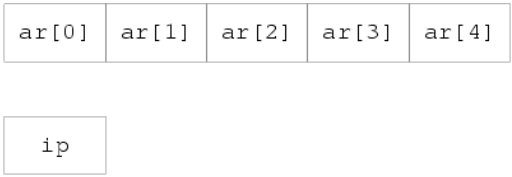

Abbildung: Schematische Darstellung

Beispiel[2]

```
1  int ar[20], *ip;
2  ip = &ar[3];
```

- Der Pointer `ip` zeigt auf das Arrayelement `ar[3]`

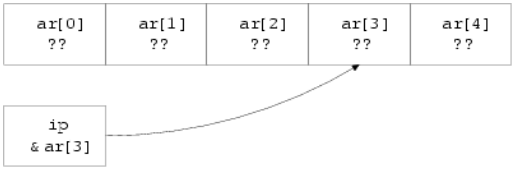

Abbildung: Schematische Darstellung

Pass by value

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

void no_change(int i) {
    i = 5;
}

int main(int argc, char** argv) {
    int a = 4;
    printf("before: a = %d\n", a);
    no_change(a);
    printf("after : a = %d\n", a);
    return 0;
}
```

```
before: a = 4
after : a = 4
```

Listing 1: Ausgabe

Pass by pointer

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

void change(int * pi) {
    *pi = 5;
}

int main(int argc, char** argv) {
    int a = 4;
    printf("before: a = %d\n", a);
    change(a);
    printf("after : a = %d\n", a);
    return 0;
}
```

```
before: a = 4
after : a = 5
```

Listing 2: Ausgabe

Beispiel

```
1 void double_swap ( double * p0 , double * p1 ) {
2     double tmp = * p0 ;
3     * p0 = * p1 ;
4     * p1 = tmp ;
5 }
```

Zeiger auf Zeiger

- Es ist möglich Zeiger auf Zeiger zu erzeugen
- Auf beliebige Datentypen anwendaber

```
1 int i = 5;
2 int * pi = &i;
3 int ** ppi = &pi;
4
5 printf("%d\n",  i);
6 printf("%d\n",  * pi);
7 printf("%d\n",  ** ppi);
```

Zeiger auf Strukturen

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

struct Point {
    double x;
    double y;
};

int main(int argc, char** argv) {
    struct Point p = {1.0, 2.5};
    struct Point * pp = &p;
    pp->x = 10;
    (*pp).y = 15;
    return 0;
}
```

void Pointer

- Kann einen beliebigen Pointertypen annehmen
- Kann einem beliegiben Pointer zugewiesen werden

NULL-Pointer

- Zeigt auf kein Objekt und ist somit ungültig
- Dereferenzierung führt meistens zu einem Laufzeitfehler
- Wird meist im Fehlerfall von den Funktionen (Konvention)

```
1 int *ip = 0;
2 int *ip = (void*) 0;
3 #include <stddef.h>
4 int *ip = NULL;
```

Das Makro NULL ist nicht vom Standard definiert und kann in verschiedene Variationen von 0 expandiert werden. Meistens sind es die Werte `0`, `0L` oder `((void*)0)`.

Einführung

Die C-Programmiersparache definiert arithmetische und Vergleichsoperatoren auf Speicheradressen. Es werden folgende Operatoren unterstützt:

Inkrement und Dekrement	`++, --`
Addition und Subtraktion	`+, -`
Vergleichsoperatoren	`<, >, <=, >=, ==, !=`

Tabelle: Liste der unterstützten Operatoren

Inkrement und Dekrement

Wie die anderen Variablen, haben die Array-Elemente Adressen

```
1 ip = &ar[5];
2 *(++ip) = 0; // ar[6] = 0
3
4 ip = &ar[5];
5 *(--ip) = 0; // ar[4] = 0
```

Addition und Subtraktion

```
1 ip = &ar[5];
2 *ip = 0; // ar[5] = 0
```

Addition von n Elementen zu dem Pointer, erzeugt einen Pointer, der n Elemente weiterzeigt

```
1 *(ip + 2) = 0; // ar[7] = 0
2 *(ip - 1) = 0; // ar[4] = 0
```

Differenz von Zeigern

```
1 ip1 = &ar[5];
2 ip2 = &ar[7];
```

Die Differenz gibt die Anzahl der Elemente zwischen den Pointern zurück

```
1 printf("ip2 - ip1 = %ld\n", (long) (ip2 - ip1));
```

Vergleich von Zeigern

- Für den Zeigervergleich können die folgende Operatoren verwendet werden
  - <, >, <=, >=, ==, !=
- Die Operatoren vergleichen Speicheradressen

Beispiel: Arrays and Pointers

```
1 for(ip = &ar[0]; ip < &ar[20];
      ↪ ip++)
2     *ip = 0;
```

```
1 for(i = 0; i < 20; i++)
2     ar[i] = 0;
```

- Der Standard sichert zu, dass auf `&ar[20]` benutzt werden kann, auch wenn dieser Element nicht exisiert
- Zugriff auf `arr[20]` ist nicht definiert

Zusammenfassung

- Pointer ist eine Variable von Typ **point-to-typ**
  - Kann eine Speicheradresse auf eine andere Variable beinhalten
  - Können verschachtelt werden
- `NULL`-Pointer ist besonders
  - Implementationsabhängig
  - Wird meist für besondere Zwecke benutzt
- Mit den Inkrement, Dekrement, Addition und Subtraktion kann man relativ zu einer Speicheradresse im Speicher navigieren
- Die Speicheradressen können mit den Verglechsoperatoren verglichen werden