
Vorlesung4

Zeiger (Grundlage)

Zeiger(Grundlage)

- Was sind Zeiger ?
- Wie definiert man Zeiger?
- Adresse und Zeiger.

Schon behandelt

-
- Verweisoperator.
 - Zeigearithmetik.
 - „Typlose Zeiger oder void Zeiger“

Vorbemerkung

- Während der dritten Vorlesung, wurde über Zeiger besprochen.
- Wir werden weiter anhand sehr einfache Beispiele lernen, wie man mit Zeigern umgeht. Diese Beispiele können Ihnen aber noch nicht zeigen, warum Zeiger so wichtig in C und C++ sind. Haben Sie bitte etwas Geduld, bis wir zu den Anwendungen von Zeigern kommen! Sie werden dann sehen, dass Sie sich ein machtvolles und in C/C++ absolut unverzichtbares Werkzeug angeeignet haben! Bis wir soweit sind (Erste Anwendung: Dynamische Speicherverwaltung !), versuchen Sie also bitte nicht, hinter unseren Beispielen irgendwelchen praktischen Nutzen zu suchen. Sie dienen wirklich nur dazu, auf möglichst einfache Weise die Bedeutung von Zeigern und den Umgang mit ihnen zu lernen.

Verweisoperator

- Der Operator * hat in einer Variablendefinition also die Bedeutung: "ist Zeiger auf (den davor stehenden Typ)". Zum Beispiel *int *p_1;*

Verweisoperator

- **Der Verweisoperator * wird von links her auf einen Zeiger angewendet, egal, ob damit Daten aus dem Speicher gelesen oder in ihn geschrieben werden sollen!**

Verweisoperator *

Was wird auf den Bildschirm nach Ausführung des Programms stehen?

```
int a;  
int *p;  
  
p = &a;  
a = 4;  
cout << "Die Ausgabe1 ist: " << p;  
//Hier liegt die Herausforderung  
cout << "Die Ausgabe2 ist: " << *p;
```

Vom Problem zum Programm

Aufgabe:

Gegeben ist das folgende Programm (Bitte nicht gleich eintippen und ausführen lassen !) :

```
1  /*****  
2  /*Da folgende Programm demonstriert den umgang mit Verweisoperator und mit Zeiger*/  
3  /*Erstellt am 20.04.07 von Dozenten FB Technik FHB*****/  
4  /* geändert am 18.04.09 um 16 Uhr*****/  
5  #include <iostream>  
6  using namespace std;  
7  int main (void)  
8  {  
9      int i, j, *pi, *pj, *p;  
10     pi = &i;  
11     pj = &j;  
12     i = 2;  
13     j = 3;  
14     *pi = *pi + *pj;  
15     p = pj;  
16     pj = pi;  
17     pi = p;  
18     *pi = 2**pj;  
19     cout <<"i = "<<i<<" , j = "<<j<<"\n";  
20     system ("Pause");  
21     return 0;  
22  
23 }
```

Vorlesung4_1.cpp

Verweisoperator *

Damit und mit einem Zeiger habe Zugriff auf die Werte in einer Adresse!

Der * Operator

Verweisoperator
Dereferenzierungsoperator

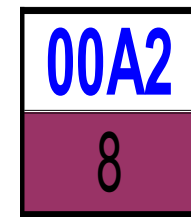
```
int main (void)
{
    int a = 5; //Deklaration und Initialisierung
    int *p; //Deklaration eines Zeigers

    p = &a; //Speichert die Adresse von a

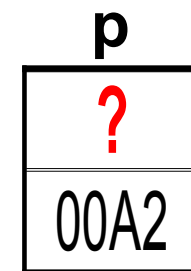
    *p = 8; //Ändert den Wert gespeichert in der
            Adresse in p

    cout << a ;
}
```

Variablen
a



Dereferenzierung



*

Der * Operator

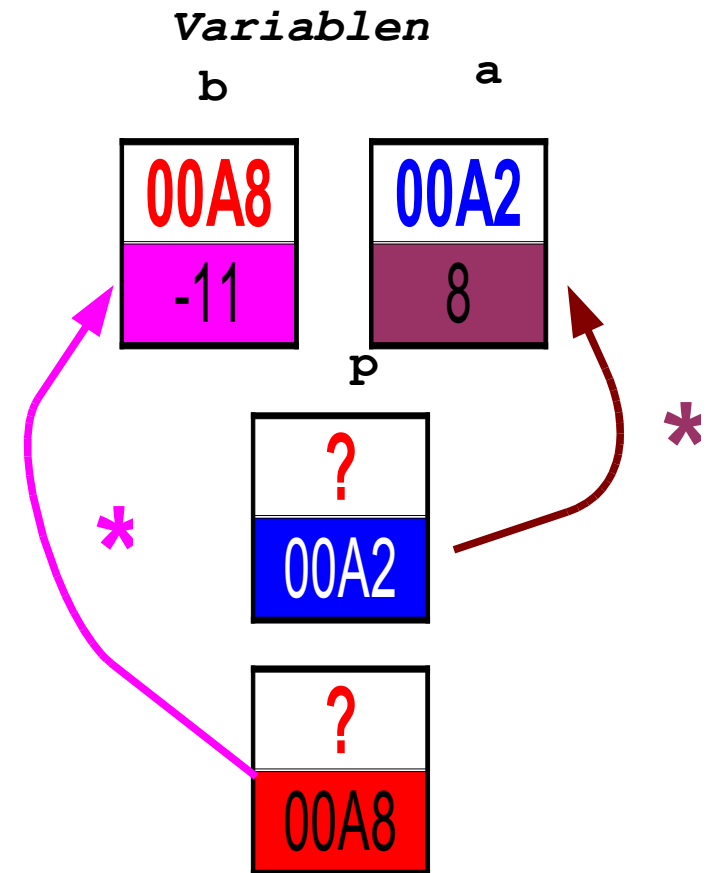
*Dereferenzierungsoperator
Verweisoperator*

```
int main (void)
{
    int a = 27, b = 34;
    int *p;

    p = &a; //Adresse von a
    *p = 8; //Ändert a!

    p = &b; //Adresse von b
    *p = -11; //Ändert b!

    cout << a << endl << b << endl;
}
```



Verweisoperator *

Je nach Tagesform nennen ihn die Leute auch

- Zugriffsoperator,
- Dereferenzierungsoperator oder
- Indirektionsoperator.
- Inhaltsoperator

Verweisoperator *

***p**

**bedeutet einen lesenden
oder schreibenden Zugriff
auf den Speicherbereich,
auf den der Zeiger p zeigt.**

Verweisoperator *



*p

bedeutet einen lesenden oder schreibenden Zugriff
auf den Speicherbereich, auf den der Zeiger `p` zeigt !

Zeigerarithmetik

Viele wichtige Anwendungen von Zeigern resultieren daraus, dass man:

- Zeigerinhalte (= Adressen !) **vergleichen** und
- mit Zeigerinhalten (Adressen) **rechnen** kann !

Die folgende Tabelle zeigt, was man mit Zeigern machen kann, d.h. Welche Operationen mit ihnen erlaubt sind.

Erlaubte Operationen mit Zeigern (p , $p1$ und $p2$ seien typgleiche Zeiger)

$++p$ bzw. $p++$

Inkrementierung

$--p$ bzw. $p--$

Dekrementierung

$p + \text{Konstante}$

Addition einer Konstanten

$p - \text{Konstante}$

Subtraktion einer Konstanten

$p1 - p2$

Subtraktion zweier Zeiger (Zeigerdifferenz)

$p1$ Vergleich $p2$

Vergleich zweier Zeiger

(Operatoren $<$, $>$, $<=$, $>=$, $==$, $!=$)

Vom Problem zum Programm

```
1 #include <iostream>
2 #include <iomanip>
3 using namespace std;
4 int main ()
5 {
6
7     int i, *pi;
8     double d, *pd;
9     i = 1;
10    d = 1.0;
11    pi = &i;
12    pd = &d;
13    cout<<"*pi = "<< *pi<<" \t"<<" *pi+1 ="<<*pi+1<<" \n";
14    cout<<"*pd = "<< *pd<<" \t"<<" *pd+1 = "<<*pd+1<<" \n";
15    cout<<"pi = "<< pi<<" \t"<<" pi+1 ="<<pi+1<<" \n";
16    cout<<"pd = "<< pd<<"\t"<<"pd+1 = "<<pd+1<<"\n";
17    system("Pause");
18    return 0;
19
20 }
21
```

vorlesung4_2.cpp

Vom Problem zum Programm

```
1 #include<iostream>
2 #include <stdio.h>
3 using namespace std;
4
5 int main ()
6 {
7     char c, *pc;
8     int i, *pi;
9     double d, *pd;
10    c = '1';
11    i = 1;
12    d = 1.0;
13    pc = &c;
14    pi = &i;
15    pd = &d;
16    printf("*pc = %c \t *pc+1 = %c \n", *pc,*pc+1);
17    printf("*pi = %d \t *pi+1 = %d \n", *pi,*pi+1);
18    printf("*pd = %lf\t *pd+1 = %lf\n\n", *pd,*pd+1);
19    printf(" pc = %p \t pc+1 = %p \n", pc, pc+1);
20    printf(" pi = %p \t pi+1 = %p \n", pi, pi+1);
21    printf(" pd = %p \t pd+1 = %p \n\n", pd, pd+1);
22    system ("Pause");
23    return 0;
24 }
25
```

vorlesung4_3.cpp

```
*pc = 1          *pc+1 = 2
*pi = 1          *pi+1 = 2
*pd = 1.000000   *pd+1 = 2.000000

pc = 0022FF77    pc+1 = 0022FF78
pi = 0022FF6C    pi+1 = 0022FF70
pd = 0022FF60    pd+1 = 0022FF68
```

Drücken Sie eine beliebige Taste . . .

Die Recheneinheit der Adress- bzw. Zeigerarithmetik ist die Speichergröße des Objekts (in Bytes), auf die die an der Operation beteiligten Zeiger zeigen!

Die Adress- bzw. Zeigerarithmetik ist also **typabhängig** !

Void-Zeiger oder „Typlose Zeiger“

Vom Problem zum Programm

```
#include <stdio.h>
#include<iostream>
using namespace std;
int main ()
{
    int *pi;
    void *pv;
    pi = (int*)0x0065FD00; // auf diese Demo-Adressen dürfen
    pv = (void*)0x0065FD04; // wir natürlich nicht zugreifen!
    pv = pi;
    pi = pv;
    cout<<"pi = "<<pi<<"\n";
    cout<<"pi = "<<pv<<"\n";
    system("Pause");
    return 0;
}
```

Betrachten Sie das folgende Programm und führen Sie das aus.
Versuchen Sie bitte, die Compiler-Fehlermeldung zu verstehen und den Fehler zu beheben!

Loesung

```
#include <stdio.h>
#include<iostream>
using namespace std;
int main ()
{
    int *pi;
    void *pv;
    pi = (int*)0x0065FD00; // auf diese Demo-Adressen dürfen
    pv = (void*)0x0065FD04; // wir natürlich nicht zugreifen!
    pv = pi;
    pi = pv;
    cout<<"pi = "<<pi<<"\n";
    cout<<"pv = "<<pv<<"\n";
    system("Pause");
    return 0;
}
```

Wir können auch "**typlose Zeiger**" definieren. Das sind Zeiger, die zwar Adressen speichern können, sich aber überhaupt nicht dafür interessieren, wessen Adressen das sind!

Man nennt solche Zeiger "**void-Zeiger**" und so definiert man sie!

-

Vorlesung4_4