
Speicheradresse und Zeiger

Inhaltsverzeichnis

- Die Speicheradresse
 - Byte
 - Variable
 - Array
- Aufgabe 1: Adressenoperator
- Aufgabe 2: Zeiger Variablen
- Aufgabe 3: Verweisoperator

Die Speicheradresse

- Jede Speicheradresse entspricht einem Byte (8 Bits) des Speicherplatzes.
- Die Speicheradresse ist **eindeutig** für jedes Byte

Byte ↔ Speicheradresse

Speicheradresse

A102

Wert

10101110

Die Speicheradresse einer Variable

- Jede Variable besitzt eine eigene Speicheradresse.
- Die Speicheradresse einer Variable ist die Adresse des **ersten Bytes** der Variable.
- Die Speicheradresse ist **eindeutig** für jede Variable.

Variable  Speicheradresse

Speicheradresse

FF02

Wert

-1023

Variablen und Speicheradresse

```
char a = 'k';
```

```
//Binär Wert: 01101011
```

```
//Adresse von "a": 02F0
```

01101011
02F0

```
short b = 292;
```

```
//Binär Wert: 00000001 00100100
```

```
//Adresse von "b": 00AB
```

00100100	00000001
00AB	00AC

```
int c = 65616;
```

```
//Binär Wert: 00000000 00000001 00000000 01010000
```

```
//Adresse von "c": FF02
```

10100000	00000000	00000001	00000000
FF02	FF03	FF04	FF05

Erster Byte



Variablen und Speicheradresse

```
int c = 65616;
```

```
//Binär Wert 00000000 00000001 00000000 01010000
```

- Das übliche Format für Intel und Windows ist ***Little-Endian:*** *niedrigste Adresse speichert das niedrigste Byte*

10100000	00000000	00000001	00000000
FF02	FF03	FF04	FF05

- Das übliche Format für Motorola-6800 ist ***Big-Endian:*** *niedrigste Adresse speichert das höchste Byte*

00000000	00000001	00000000	10100000
FF02	FF03	FF04	FF05

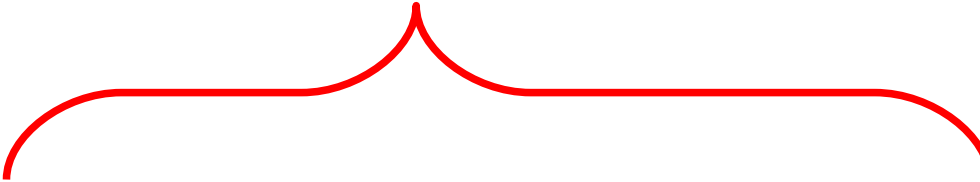
Arrays und Speicheradresse

- Jede Position des Arrays hat eine eigene Speicheradresse.
- Die Speicheradresse des Arrays ist die Adresse der **ersten Position (Position NULL)**.
- Die Speicheradressen sind aufsteigend in der ersten Position beginnend mit Schrittweite die Größe des Datentyps (int oder float oder double...) .

Arrays und Speicheradresse

```
int a[7] = {65616, 2, 0, -872, 1009, 2191, -7427}
```

WERTE	65616	2	0	-872	1009	2191	-7427
ADRESSE	FF02	FF06	FF0A	FF0E	FF12	FF16	FF1A
POSITION	0	1	2	3	4	5	6



10100000	00000000	00000001	00000000
FF02	FF03	FF04	FF05

Bytebelegung der ersten Position

Adresse der 1. Position: FF02
+4 Bytes (int)
Adresse der 2. Position: FF06
+4 Bytes (int)
Adresse der 3. Position: FF0A
u.s.w.

Aufgabe 1: der Adressenoperator

Variablen und Adressen: Aufgabe 1

Eine einfache Aufgabe:

Es soll ein Programm erstellt werden.

Das Programm sollte

- Variablen von verschiedenen Typen einlesen und
- deren Werte sowie **deren Adressen** auf dem Bildschirm ausgeben.

Bevor wir das implementieren können, müssen wir neue Begriffe lernen.

Aufgabe1: Lösungsweg

- Einlesen von Variablen. **Sehr einfach**
- Adressen von Variablen ermitteln (Adressenoperator &) und ausgeben. **Wie?**

Ermittlung der Adresse

Wie kann man die Adresse einer Variable ermitteln?

Durch den **Operator &**

&Variablenname

AdresseArrayVariable.cpp

Ermittlung der Adresse

Wie kann man die Adresse eines Arrays ermitteln?

Durch den **Namen des Arrays**

Arrayname

Äquivalent:

Durch den **Adressen der ersten Position**

&Arrayname[0]

Aufgabe1: Lösung

- Einlesen von Variablen. **Sehr einfach**
- Adressen von Variablen ermitteln (Adressenoperator &) und ausgeben. **Wie? Mit dem Adressenoperator**

Lösungsdatei: *DatentypenAdressen.cpp*

Aufgabe 2: die Zeiger-Variablen

Zeiger und Adressen: Aufgabe 2

Eine im Moment nicht einfache Aufgabe:

Es sollte ein Programm entwickelt werden.

Das Programm sollte:

- Variablen von unterschiedlichen Datentypen einlesen
- deren Adressen in Variablen **speichern** und schließlich ausgeben

Aufgabe 2: Lösungsweg

- Einlesen von Variablen. **Sehr einfach**
- Adressen von Variablen Speichern: **Wo?**
- Gespeicherte Adresse ausgeben. **Sehr einfach**

Bevor wir das implementieren können, müssen wir neue Begriffe lernen.

Adressen: Ein neuer Datentyp

3 wichtige Aussagen über Speicheradressen in C\C++

- Das Ergebnis des Adressenoperators (&) ist ein neuer Datentyp und zwar eine Speicheradresse.
- Eine Speicheradresse kann nicht direkt in einer **ganzzahligen** Variable (`short`, `int`, `long`, `long long`...) gespeichert werden
- Es gibt einen neuen Datentyp für das Speichern der Adressen: **Zeiger-Datentyp**.

Was sind Zeiger ?

- Nichts, wovor man sich fürchten muss!
Zeiger (engl. **Pointer**) sind **ganz gewöhnliche Variablen**.
- Anders als Variablen vom Typ `char`,
`int`, `long`, `float`, `double` usw. wollen Zeiger
jedoch **Adressen speichern!**

Vereinbarung von Zeiger

Deklaration

```
int    *p_1;    //Zeiger von int
double *p_2;    //Zeiger von double
char   *p_3;    //Zeiger von char
short  *p_4;    //zeiger von short
```

Vereinbarung von Zeiger

Allgemeine Deklarationsyntax

- Allgemein wird also eine Zeigervariable (kurz: ein Zeiger) so deklariert:

Typ **p*;

Man sagt: "*p* ist ein Zeiger auf eine Variable vom Datentyp *Typ*."

Vereinbarung von Zeiger

- Der Typ eines Zeigers ist: `Typ *` (oder ohne Leerzeichen: `Typ*`)

Nach dem vorangegangenen Beispiel

- p_1 hat den Typ `int*`
- p_2 hat den Typ `double*`
- p_3 hat den Typ `char*`

Lösung: Aufgabe 2

Mit den neuen Datentyp **Zeigern** können wir nun der zweiten Schritt implementieren.

- Einlesen von Variablen. **Sehr einfach**
- Adressen von Variablen Speichern: **Wo? In Zeigervariablen**
- Gespeicherte Adresse ausgeben. **Sehr einfach**

Lösungsdatei: *DatentypenAdressen.cpp*

Aufgabe 3: der Verweisoperator

Zeiger und Adressen: Aufgabe 3

Eine letzte fordernde Aufgabe:

Es sollte ein Programm entwickelt werden ,das Programm sollte:

- Variablen von unterschiedlichen Datentypen einlesen
- deren Adressen in Variablen speichern und
- letztendlich deren **Inhalt aus der Adresse her** auf dem Bildschirm ausgeben.

Bevor wir das implementieren können, müssen wir neue Begriffe lernen.

Verweisoperator *

Ein neuer Operator (unär = wirkt auf einen einzelnen Ausdruck, wie das Minus-Operator)

Nicht zu verwechseln mit dem Multiplikation-Operator (binär = wirkt auf zwei Ausdrücke)

Beispielcode:

```
//Dereferenzierung1.cpp
```

```
int a;
```

```
int *p;
```

```
p = &a;
```

```
a = -1023;
```

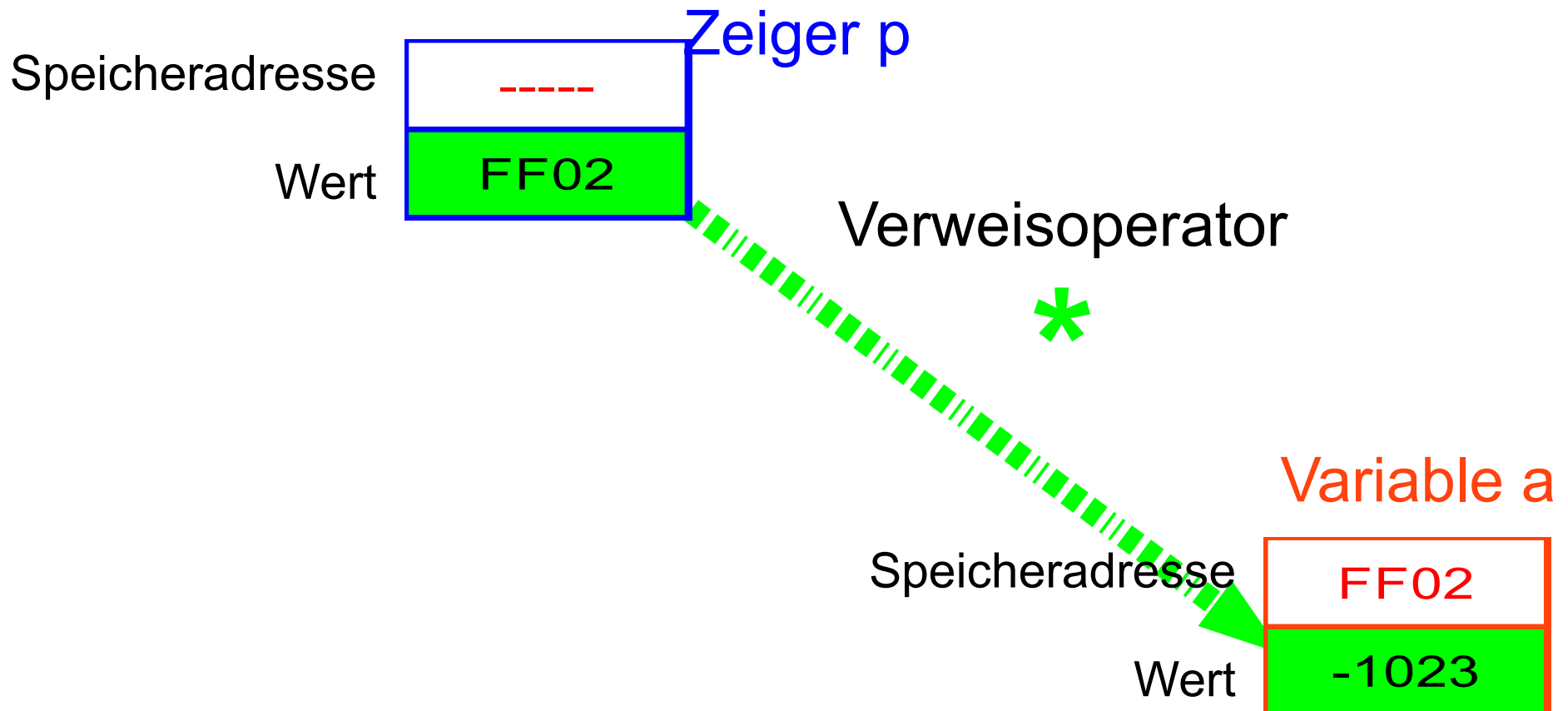
```
cout << "Die Ausgabe1 ist: " << p;
```

```
//Hier wirkt der Verweisoperator auf den Zeiger p
```

```
cout << "Die Ausgabe2 ist: " << *p;
```

Verweisoperator *

Mit dem Verweisoperator und mit einem Zeiger hat man **Zugriff auf den Wert** in einer Adresse!



Der * Operator

Verweisoperator Dereferenzierungsoperator

//Dereferenzierung2.cpp

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

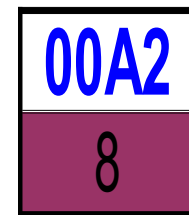
    p = &a; //Speichert die Adresse von a

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

    cout << a ;
}
```

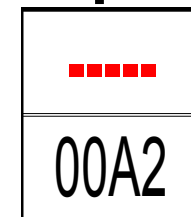
Variablen

a



Dereferenzierung

p



*

Lösung: Aufgabe 3

Mit dem neuen Operator **Verweisoperator** können wir Momentan den dritten Schritt der Aufgabe implementieren.

- Einlesen von Variablen. **Sehr einfach**
- Adressen von Variablen Speichern: **Wo? In Zeigervariablen**
- Inhalt der Variablen **aus der Adresse her** ausgeben. **Wie? Mit dem Verweisoperator**

Lösung: `DatentypenVerweisoperator.cpp`