Übersicht

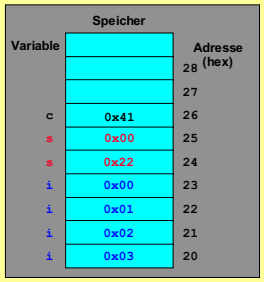
- ✓Vergleichs-Operatoren und Boolsche Werte
- ✓Verzweigung mit if / else und switch
- ✓Schleifen
- ✓Präprozessordirektiven
- ✓Funktionen
- • **Zeiger und Adressen**
- • Arrays (Felder)
- • Strukturen

Adressen

- Jeder Variablen wird vom Compiler eine **Speicher-Adresse** zugewiesen (*auch jeder Funktion*).
- Eine **Variable** repräsentiert den **Inhalt** einer oder mehrerer Speicherzellen (*Bytes*) ab dieser Adresse.
  - Die **Bedeutung** der Speicherzellen wird bestimmt durch den **Datentyp** der Variablen (*Interpretation des Inhalts*).
- Der Zugriff auf den **Inhalt** der Speicherzellen erfolgt über den **Variablen-Namen**.

Speicherbelegung von Variablen

```
1   int   i;
2   short s;
3   char  c
4
5   main()
6   {
7       ...
8       i = 66051;
9       s = 34;
10      c = 'A';
11      ...
12  }
```



Zeiger

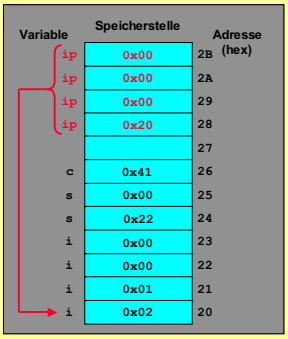
- **"Zeiger" (pointer)** = Startadresse eines Speicherbereichs.
  - Die **Größe** dieses Speicherbereichs ist zunächst unklar.
- **Zeiger** können in **Zeiger-Variablen** abgelegt werden.
  - Die **Größe** ist meist **4 Byte / 32 Bit**.
  - Der **Inhalt** wird als **Adresse** interpretiert.
  - Bei der Definition wird ihr ein **Datentyp** zugeordnet.
  - Sie kann *nur* auf Daten diesen Typs zeigen.
  - Der **Datentyp** bestimmt, wieviele Byte dem Zeiger zugeordnet werden und wie sie zu interpretieren sind.

Zeiger

- **Zeiger-Operatoren**:
  - **&** = **"Adresse von"**
  - ***** = **"Wert am Ziel von"** (*Dereferenzierung*)
  - ***** = **"Zeiger auf"** (*Variablen-Definition*)
- **Achtung**: Doppeltbelegung von **&** und *****:
  - **&** = Bitweise UND-Verknüpfung
  - ***** = Multiplikation
  - – Welche Bedeutung jeweils gilt, wird durch den **Operator-Vorrang** entschieden.

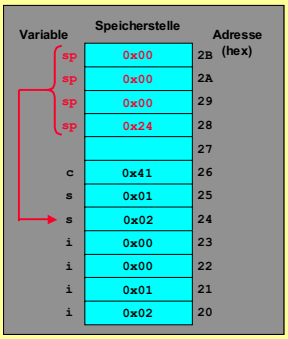
Zugriff auf Variablen per Zeiger

```
1   int   i;
2   short s;
3   char  c;
4   int*  ip;
5
6   main()
7   {
8       ...
9       ip = &i;
10      *ip = 258;
11      ...
12  }
```



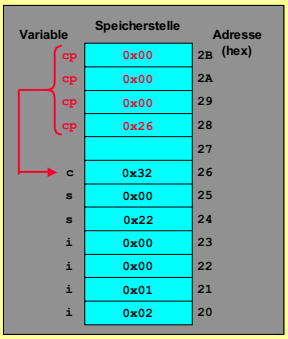
Zugriff auf Variablen per Zeiger

```
1  int   i;
2  short  s;
3  char   c;
4  short* sp;
5
6  main()
7  {
8     ...
9     sp = &s;
10    *sp = 258;
11    ...
12 }
```



Zugriff auf Variablen per Zeiger

```
1  int   i;
2  short s;
3  char  c;
4  char* cp;
5
6  main()
7  {
8     ...
9     cp = &c;
10    *cp = '2';
11    ...
12 }
```



Basisoperationen mit Zeigern

- Definition einer Zeiger-Variable:
  - Legt fest, auf welchen **Datentyp** sie zeigt.
  - Beispiel: `int* ip;` `float* fp;`
- Welche **Schreibweise** ist die richtige?

"`*ip` *ist ein* `int`":	`int *ip;`
"`ip` *ist ein Zeiger auf* `int`"	`int* ip;`
Weder noch (*vermeiden*):	`int * ip;`

  - **Tipp**: Suchen Sie sich diejenige Schreibweise heraus, die Sie am besten verstehen, aber bleiben Sie dann auch dabei.

Basisoperationen mit Zeigern

- Zuweisung eines Wertes an eine Zeiger-Variable per:

Initialisierung:	`int* ip = &i;`
Zuweisung einer Adresse:	`fp = &f;`
Zuweisung einer Konstanten:	`fp = NULL;`

- `NULL` ist der "**unmögliche Zeiger**" und eigentlich nur eine klarere Schreibweise für den Wert `0`.
  - Eine Zeiger-Variable mit diesem Wert kann **nie** auf eine gültige Speicherstelle zeigen.
  - Zeiger-Variablen werden damit meist **initialisiert**.

Basisoperationen mit Zeigern

- Speichern eines Wertes an der Stelle, auf die eine Zeiger-Variable zeigt ("**Dereferenzierung**"):

```
*ip = 123;
```

- Holen eines Wertes von der Stelle, auf die eine Zeiger-Variable zeigt ("**Dereferenzierung**"):

```
printf("Wert %d\n", *ip);
```


Basisoperationen mit Zeigern

- Den **Wert** einer Zeiger-Variablen ausgeben (*die Adresse, auf die sie zeigt,* **pointer**):

```
printf("Wert %p\n", ip);
```

  - Meist in **Hexadezimal-Form** ausgegeben (evtl. durch "`:`" in 2 Teile getrennt).
- Die **Adresse** einer Variablen an eine Fkt. übergeben:

```
int zahl;
scanf(" %d", &zahl);
```

  - `scanf` benötigt nicht den Wert der Variablen sondern ihre **Adresse**, um dort einen Wert ablegen zu können.

Weitere Zeigeroperationen

- Zeiger können (fast) wie **Zahlen** behandelt werden:
  - Eine ganze Zahl darf zu ihnen **addiert** oder von ihnen **subtrahiert** werden.
  - Sie dürfen **miteinander verglichen** werden.
- Dabei ist die "**Einheit**", um die ein Zeiger verändert wird, abhängig vom **Datentyp**, auf den er zeigt:
  - Zeiger vom Datentyp `char` ist ein **Vielfaches von 1**
  - Zeiger vom Datentyp `short` ist ein **Vielfaches von 2**.
  - Zeiger vom Datentyp `long` ist ein **Vielfaches von 4**.
  - Zeiger vom Datentyp `double` ist ein **Vielfaches von 8**.

Weitere Zeigeroperationen

- **Arithmetische Operationen** mit Zeigern:
  - Um 2 Einheiten "vorschieben": `ip = ip + 2;`
  - Um 5 Einheiten "zurückschieben": `ip = ip - 5;`
  - Anzahl Einheiten dazwischen: `n = ip1 - ip2;`
- **Vergleich** von Zeigern:
  - "ungleich" ungültiger Zeiger: `ip != NULL`
  - "gleich" ungültiger Zeiger: `ip == NULL`
  - Zeiger `ip1` "vor" Zeiger `ip2`: `ip1 < ip2`
  - Zeiger `ip1` "nach od. gleich" `ip2`: `ip1 >= ip2`

Weitere Zeigeroperationen

- **Arithmetische Operationen** mit Zeigern:
  - Um 2 Einheiten "vorschieben": `ip += 2;`
  - Um 5 Einheiten "zurückschieben": `ip -= 5;`
- **Inkrement** / **Dekrement** von Zeigern:
  - Zeiger um 1 Einheit "vorschieben": `++ip;`
  - Zeiger um 1 Einheit "vorschieben": `ip++;`
  - Zeiger um 1 Einheit "zurückschieben": `--ip;`
  - Zeiger um 1 Einheit "zurückschieben": `ip--;`

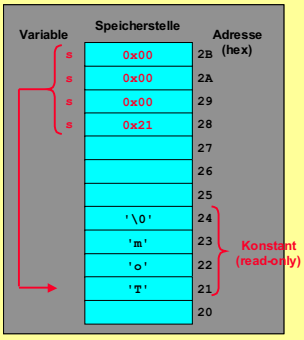
Zeiger auf Strings

- Eine **Zeichenkette** (String) hat als "**Wert**" die Adresse des 1. Zeichens im String.
  - Dieser Wert ist ein sogenannter "**character pointer**".
  - Die Startadresse von Strings kann daher in Variablen des Typs `char*` gespeichert werden.
- Zeichenketten-Variablen-Definition:

String-Array (Feld):	`char s[] = "tom";`
String-Array (Feld):	`char s[4] = "tom";`
Zeiger-Variable auf String:	`char* s = "tom";`

Speicherung einer Zeiger-Variablen auf String

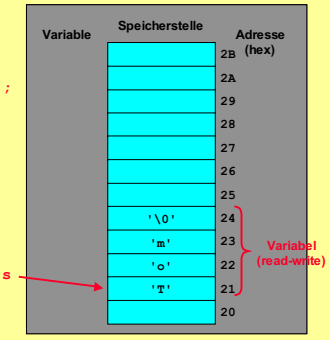
```
1  char* s = "Tom";
2
3  main()
4  {
5     ...
6     printf("%s\n",
7        s);
8     s[0] = "t";
9     ...
   }
```



Speicherung eines String-Arrays

```
1  char s[] = "Tom";
2  char s[] = { 'T',
3    'o', 'm', '\0' };
4
5  main()
6  {
7     ...
8     printf("%s\n",
9        s);
      s[0] = "t";
      ...
   }
```

s ist ein "**Alias**" (*anderer Name*) für die Start-Adresse von "Tom"



Verwendung von String-Zeigern

charptr.c

```
1  char* text = "Dies ist ein langer Text als Platzhalter";
2  char* cp;           // cp = character pointer
3  int   laenge = 0;
4
5  printf("Bitte Text eingeben: ");
6  scanf("%s", text);   // KEIN & nötig, da bereits Zeiger!
7
8  // 1. Methode: Textlänge ermitteln
9  for (cp = text; *cp != '\0'; ++cp)
10     ++laenge;
11 // 2. Methode: (unter Ausnutzung der Schleife vorher)
12 for (cp = text; *cp != '\0'; ++cp)
13     ;  /* nix zu tun, nur Zeiger bis Nullbyte schieben */
14 laenge = cp - text;
15
16 printf("%d Zeichen eingegeben!\n", laenge);
17
18 // Text mit Blank zwischen jedem Zeichen ausgeben
19 for (cp = text; *cp != '\0'; ++cp)
20     printf("%c ", *cp);
```

Zugriff auf einzelne Speicherzellen

- Mit Zeiger-Variablen vom Typ **`char*`** kann auf **einzelne Bytes** im Speicher zugegriffen werden.

```
char* cp = 0x0000FFFF;
printf("%02d\n", *cp);
```

- Zeiger-Typen können per **cast** ineinander umgewandelt werden (*schön, aber gefährlich!*):

```
int   i;
int*  ip = &i;
char* cp;
cp = (char*)ip;   oder   cp = (char*)&i;
```


Zeiger als Funktionsparameter

- An eine Funktion kann anstelle des **Wertes** einer Variablen ihre **Adresse** übergeben werden:

```
Übergabe des Wertes von i:     func1(i);
Übergabe der Adresse von i:    func2(&i);
```

- Mit Variante 1 hat sie nur "**lesenden**" Zugriff.
  - Eigentlich wird der **Wert** der Variablen in eine neue (*lokale*) Variable **kopiert** (**call by value**).
- Mit Variante 2 hat sie auch "**schreibenden**" Zugriff.
  - Eigentlich wird die **Adresse** der Variablen in eine neue (*lokale*) Variable **kopiert** (**call by reference**).

Zeiger als Funktionsparameter

- Die formalen Parameter der Funktion müssen dann als **Zeiger-Variable** definiert werden.

```
func1(int i);
func2(int* i);
```

- In der Funktion muß bei Zugriff auf die Variable immer "**dereferenziert**" werden.

```
*i = 100;      printf("%d\n", *i);
```

***Strings** und **Arrays** werden grundsätzlich per Adresse an eine Funktion übergeben.*

Verwendung von Zeigern als Funktionsparameter

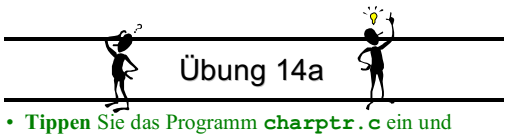
swap.c 1.Teil

```
1  #include <stdio.h>
2
3  void swap1(int*, int*);  // Prototyp
4  void swap2(int, int);    // Prototyp
5
6  int main()
7  {
8      int z1, z2;
9
10     printf("Bitte 2 Zahlen eingeben: ");
11     scanf(" %d %d", &z1, &z2);
12
13     printf("z1: %d, z2: %d\n", z1, z2);
14     swap1(&z1, &z2);                  // by-reference
15     printf("z1: %d, z2: %d\n", z1, z2);
16     swap2(z1, z2);                    // by-value
17     printf("z1: %d, z2: %d\n", z1, z2);
18
19     return 0;
20 }
```

Verwendung von Zeigern als Funktionsparameter

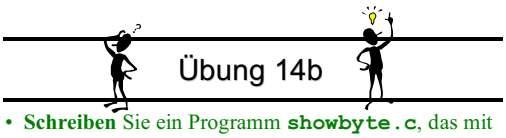
swap.c 2.Teil

```
21 // Vertauschen mit "call-by-reference"
22 void swap1(int* pZahl1, int* pZahl2)
23 {
24    int temp;
25
26    temp    = *pZahl1;
27    *pZahl1 = *pZahl2;
28    *pZahl2 = temp;
29 }
30
31 // Vertauschen mit "call-by-value"
32 void swap2(int Zahl1, int Zahl2)
33 {
34    int temp;
35
36    temp  = Zahl1;
37    Zahl1 = Zahl2;
38    Zahl2 = temp;
39 }
```



Übung 14a

- **Tippen** Sie das Programm **`charptr.c`** ein und testen Sie seine Funktionalität.
- **Tippen** Sie das Programm **`swap.c`** ein und testen Sie seine Funktionalität.
- **Schreiben** Sie ein Programm **`pointer.c`**, das die Verwendung von **Zeigern** zeigt.



Übung 14b

- **Schreiben** Sie ein Programm **`showbyte.c`**, das mit einem **`char*`** auf die einzelnen Bytes von **`short`**, **`int`**, **`long`**, **`float`** und **`double`**-Variablen zugreift und diese als **Byte-Code** und als **Zeichen** ausgibt.
  - **Überprüfen** Sie damit, wie die **`0`**, **`1`**, **`1.0`**, **`-1.0`** in den verschiedenen Datentypen codiert werden?
  - **Ergänzen** Sie eine Variable **`s`** vom Typ **`char*`**, die auf den Text "**`Hello`**" zeigt und geben Sie die **Einzel-Bytes** dieses Strings durch Zugriff über eine Zeiger-Variable ebenfalls aus.