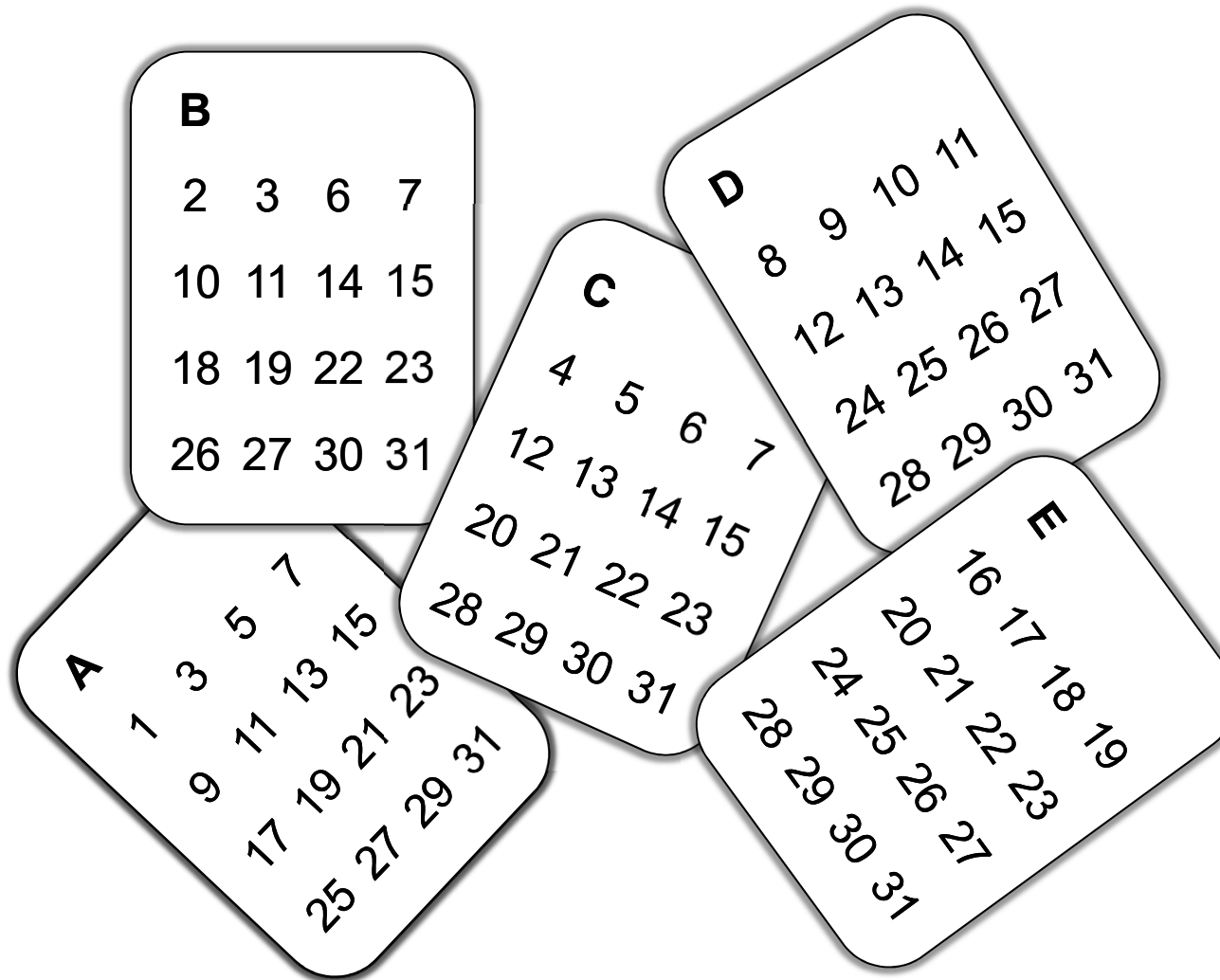


Kapitel 6

Elementare Datentypen und ihre Darstellung

Zahlenraten

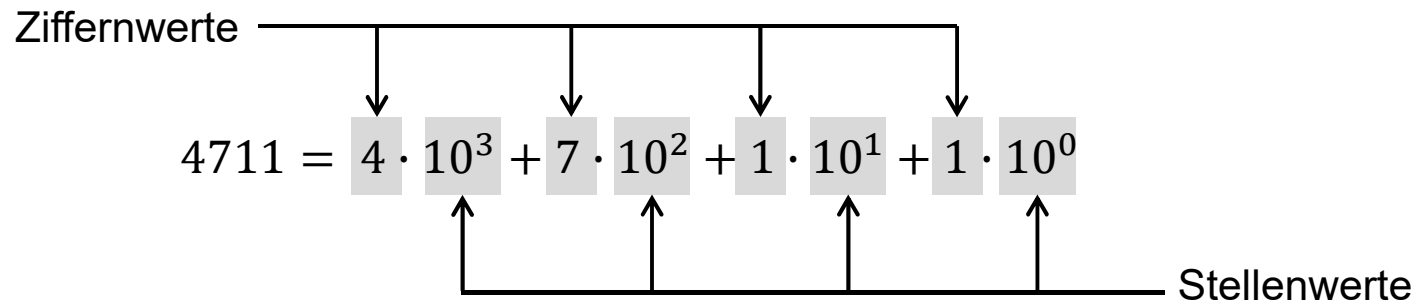


Denken Sie sich eine Zahl zwischen 0 und 32 ...

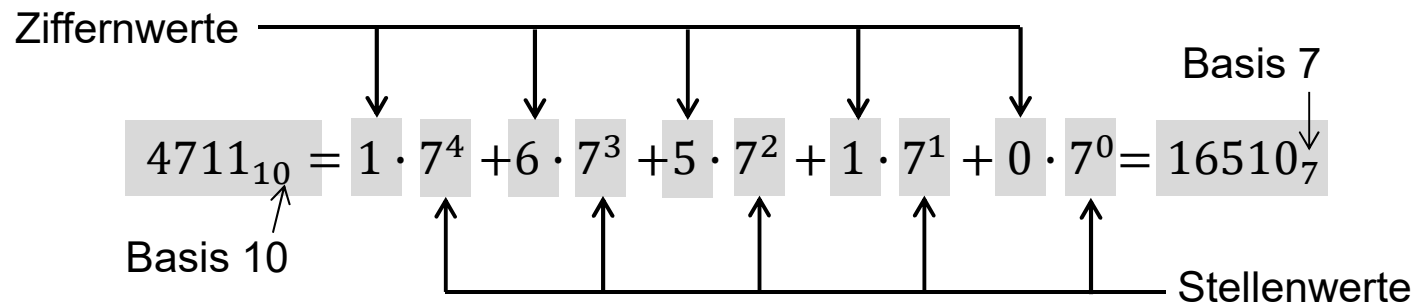
Zahlen sind abstrakte mathematische Objekte. Damit man sie konkret benutzen kann, brauchen sie eine "Benutzerschnittstelle" mittels derer man sie addieren, multiplizieren oder vergleichen kann.



Zur Darstellung von Zahlen verwendet man sogenannte Stellenwertsysteme. Im Alltag verwenden wir das Dezimalsystem:



Diese Darstellung beruht auf der Zahl 10 als Basis. Man kann jede andere natürliche Zahl größer als 1 als Basis verwenden. Man erhält dann nur eine andere Ziffernfolge:



Wie kommt man zu dieser Darstellung?

Umrechnung: vom 10er ins 7er-System

$$4711_{10} = 16510_7 = 1 \cdot 7^4 + 6 \cdot 7^3 + 5 \cdot 7^2 + 1 \cdot 7^1 + 0 \cdot 7^0$$

$$= \left(\left((1 \cdot 7 + 6) \cdot 7 + 5 \right) \cdot 7 + 1 \right) \cdot 7 + 0$$

Man erhält die Ziffernwerte im 7er-System, wenn man die Reste bei Division durch 7 betrachtet:

$$\begin{array}{rcl}
 4711 & = & 673 \cdot 7 + 0 \\
 673 & = & 96 \cdot 7 + 1 \\
 96 & = & 13 \cdot 7 + 5 \\
 13 & = & 1 \cdot 7 + 6 \\
 1 & = & 0 \cdot 7 + 1
 \end{array}$$

1 6 5 1 0

Im 7er-System ist alles wie im 10er-System, außer, dass es nur die Ziffern 0-6 gibt und dass beim Zählen immer bei 6 ein Übertrag erfolgt.

| | | | | | | | | | | | | | | | | | |
|-------------|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|
| 10er-System | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 7er-System | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 20 | 21 | 22 |

Das 7er-System ist zum Rechnen genauso geeignet, wie das 10er-System. Die scheinbare Überlegenheit des 10er-Systems liegt in der Vertrautheit mit diesem System. Hätte der Mensch 12 Finger, würden wir heute wahrscheinlich im 12er-System rechnen.

Umrechnung durch ein C-Programm

```
void main()
{
    int basis = 7;
    int zahl = 4711;
    int z;

    printf( "Basis: %d\n", basis);
    printf( "-----\n");

    for( z = zahl ; z != 0; z = z/basis)
        printf( "%4d = %4d*%d + %2d\n", z, z/basis, basis, z%basis);

    printf( " -----\n\n");
}
```

Die Basis ist 7, kann aber geändert werden.

Die Zahl wird so lange durch die Basis geteilt, bis nichts mehr übrig bleibt.

Hier wird eine Feldbreite für die Ausgabe festgelegt.

Hier wird jeweils eine Zeile ausgegeben.

| Basis: 7 | | | |
|----------|---|-------|-----|
| 4711 | = | 673*7 | + 0 |
| 673 | = | 96*7 | + 1 |
| 96 | = | 13*7 | + 5 |
| 13 | = | 1*7 | + 6 |
| 1 | = | 0*7 | + 1 |

Die gesuchte Ziffernfolge ergibt sich dann von unten nach oben: 16510

Das Programm zeigt noch einmal die Bedeutung von Integer-Division und Modulo-Operation.

Umrechnung für andere Basen

| Basis: 2 | | |
|----------|---|------------|
| 4711 | = | 2355*2 + 1 |
| 2355 | = | 1177*2 + 1 |
| 1177 | = | 588*2 + 1 |
| 588 | = | 294*2 + 0 |
| 294 | = | 147*2 + 0 |
| 147 | = | 73*2 + 1 |
| 73 | = | 36*2 + 1 |
| 36 | = | 18*2 + 0 |
| 18 | = | 9*2 + 0 |
| 9 | = | 4*2 + 1 |
| 4 | = | 2*2 + 0 |
| 2 | = | 1*2 + 0 |
| 1 | = | 0*2 + 1 |

| Basis: 3 | | |
|----------|---|------------|
| 4711 | = | 1570*3 + 1 |
| 1570 | = | 523*3 + 1 |
| 523 | = | 174*3 + 1 |
| 174 | = | 58*3 + 0 |
| 58 | = | 19*3 + 1 |
| 19 | = | 6*3 + 1 |
| 6 | = | 2*3 + 0 |
| 2 | = | 0*3 + 2 |

| Basis: 4 | | |
|----------|---|------------|
| 4711 | = | 1177*4 + 3 |
| 1177 | = | 294*4 + 1 |
| 294 | = | 73*4 + 2 |
| 73 | = | 18*4 + 1 |
| 18 | = | 4*4 + 2 |
| 4 | = | 1*4 + 0 |
| 1 | = | 0*4 + 1 |

| Basis: 5 | | |
|----------|---|-----------|
| 4711 | = | 942*5 + 1 |
| 942 | = | 188*5 + 2 |
| 188 | = | 37*5 + 3 |
| 37 | = | 7*5 + 2 |
| 7 | = | 1*5 + 2 |
| 1 | = | 0*5 + 1 |

| Basis: 6 | | |
|----------|---|-----------|
| 4711 | = | 785*6 + 1 |
| 785 | = | 130*6 + 5 |
| 130 | = | 21*6 + 4 |
| 21 | = | 3*6 + 3 |
| 3 | = | 0*6 + 3 |

| Basis: 7 | | |
|----------|---|-----------|
| 4711 | = | 673*7 + 0 |
| 673 | = | 96*7 + 1 |
| 96 | = | 13*7 + 5 |
| 13 | = | 1*7 + 6 |
| 1 | = | 0*7 + 1 |

| Basis: 8 | | |
|----------|---|-----------|
| 4711 | = | 588*8 + 7 |
| 588 | = | 73*8 + 4 |
| 73 | = | 9*8 + 1 |
| 9 | = | 1*8 + 1 |
| 1 | = | 0*8 + 1 |

| Basis: 9 | | |
|----------|---|-----------|
| 4711 | = | 523*9 + 4 |
| 523 | = | 58*9 + 1 |
| 58 | = | 6*9 + 4 |
| 6 | = | 0*9 + 6 |

Ist die Basis größer als 10, ergeben sich unter Umständen Ziffernwerte ≥ 10 :

| Basis: 13 | | |
|-----------|---|------------|
| 4711 | = | 362*13 + 5 |
| 362 | = | 27*13 + 11 |
| 27 | = | 2*13 + 1 |
| 2 | = | 0*13 + 2 |

Zur Darstellung dieser Ziffernwerte verwendet man dann die Ziffernsymbole A oder a (für 10), B oder b (für 11), C oder c (für 12) usw.:

$$4711_{10} = 21B5_{13} = 21b5_{13}$$

Warum interessiert uns das Dualsystem?

Ein Digitalrechner kennt intern nur zwei Zustände, die mit 0 und 1 bezeichnet werden. Alle Daten und auch Programme im Rechner bestehen aus Folgen von 0 und 1.

Der Rechner braucht eine Organisation, über die er effizient auf die Daten und Programme zugreifen kann. Dazu wird der Speicher des Rechners in einzelne Speicherzellen unterteilt und die Speicherzellen werden fortlaufend nummeriert.

Intern sind die Nummern der Speicherzellen und auch die Werte in den Speicherzellen durch Folgen von 0 und 1, also durch Zahlen im Zweiersystem (Dualsystem) gegeben.

| Speicherzelle | | | | | Wert | | | | | | | |
|---------------|---|---|---|---|------|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 0 | | | | | | | | |
| 3 | 0 | 0 | 1 | 1 | | | | | | | | |
| 4 | 0 | 1 | 0 | 0 | | | | | | | | |
| 5 | 0 | 1 | 0 | 1 | | | | | | | | |
| 6 | 0 | 1 | 1 | 0 | | | | | | | | |
| 7 | 0 | 1 | 1 | 1 | | | | | | | | |
| 8 | 1 | 0 | 0 | 0 | | | | | | | | |
| 9 | 1 | 0 | 0 | 1 | | | | | | | | |
| 10 | 1 | 0 | 1 | 0 | | | | | | | | |
| 11 | 1 | 0 | 1 | 1 | | | | | | | | |
| 12 | 1 | 1 | 0 | 0 | | | | | | | | |
| 13 | 1 | 1 | 0 | 1 | | | | | | | | |
| 14 | 1 | 1 | 1 | 0 | | | | | | | | |
| 15 | 1 | 1 | 1 | 1 | | | | | | | | |

weitere
Werte





Warum interessiert uns ein anderes Zahlensystem als das Dualsystem?

Stellen Sie sich vor, dass wir im Dualsystem rechnen würden und einen Laptop kaufen wollten:

Für uns Menschen hat das Dualsystem erhebliche Nachteile.

Wegen der kleinen Basis sind die Zahlen viel zu lang und nur umständlich zu handhaben.

Außerdem fehlt uns jegliche Größenvorstellung für Dualzahlen.

| Rang | Produkt | Preis |
|------|--|------------------------|
| 1. | Fujitsu Lifebook E743 (E7430M55A1DE)  | ab 1110100011 € |
| 2. | Fujitsu Lifebook E753 (E7530M55A1DE)  | ab 1110011001 € |
| 3. | Acer Aspire V5-573G-54208G50akk (NX.MCEE.005)  | ab 1010111011 € |
| 4. | Samsung Serie 4 400B5C (A04DE)  | ab 1101111100 € |

Gesucht ist daher ein Zahlensystem, dessen Basis näher an der vertrauten Basis 10 liegt und einfache Umrechnungen ins Dualsystem erlaubt.

Das Oktalsystem (Basis 8)

Wir betrachten eine Zahl im Dualsystem und bilden 3-er Gruppen:

$$\begin{aligned}
 101\ 100\ 011_2 &= (1 \cdot 2^8 + 0 \cdot 2^7 + 1 \cdot 2^6) + (1 \cdot 2^5 + 0 \cdot 2^4 + 0 \cdot 2^3) + (0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) \\
 &= (1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0) \cdot 2^6 + (1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0) \cdot 2^3 + (0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0) \cdot 2^0 \\
 &= 5 \cdot 2^6 + 4 \cdot 2^3 + 3 \cdot 2^0 \\
 &= 5 \cdot 8^2 + 4 \cdot 8^1 + 3 \cdot 8^0 \\
 &= 543_8
 \end{aligned}$$

Es folgt:

Zwischen Oktal- und Dualdarstellung kann man "ziffernweise" mit folgender Tabelle

| Oktal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|-------|-----|-----|-----|-----|-----|-----|-----|-----|
| Dual | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |

umrechnen. In der Dualdarstellung sind dabei von rechts her Dreiergruppen zu bilden:

| Oktal | 5 | | | 6 | | | 4 | | | 3 | | |
|-------|---|---|---|---|---|---|---|---|---|---|---|---|
| Dual | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

Die einfache Umrechnung hat ihren Grund darin, dass 8 eine Potenz von 2 ist. Dreiergruppen werden gebildet, weil $8 = 2^3$ ist.

Das Hexadezimalsystem (Basis 16)

Das zum Oktalsystem Gesagte gilt analog für das Hexadezimalsystem, da 16 ebenfalls eine Potenz von 2 ist ($16 = 2^4$).

Zwischen Hexadezimal- und Dualdarstellung kann man "ziffernweise" mit folgender Tabelle

| Hexadezimal | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | a | b | c | d | e | f |
|-------------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| Dual | 0000 | 0001 | 0010 | 0011 | 0100 | 0101 | 0110 | 0111 | 1000 | 1001 | 1010 | 1011 | 1100 | 1101 | 1110 | 1111 |

umrechnen. In der Dualdarstellung sind dabei von rechts her Vierergruppen zu bilden:

| Dual | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
|-------------|---|---|---|---|---|---|---|---|---|---|---|---|
| Hexadezimal | b | | | | a | | | | 3 | | | |

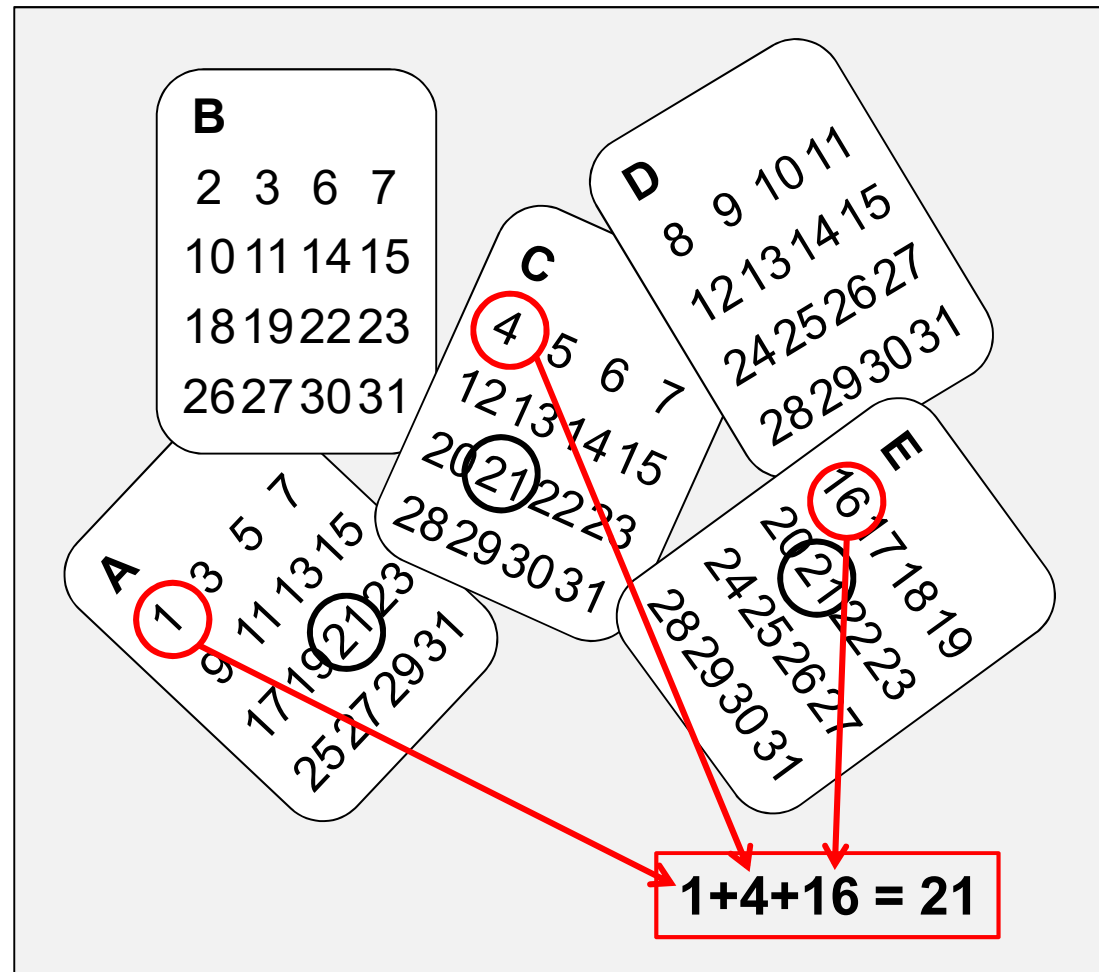
Zur Umformung zwischen Dual-, Oktal- und Hexadezimalsystem und umgekehrt muss man also nicht rechnen:

| Oktal | 5 | | | 6 | | | 4 | | | 3 | | |
|-------------|---|---|---|---|---|---|---|---|---|---|---|---|
| Dual | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| Hexadezimal | b | | | | a | | | | 3 | | | |

Oktal- und Hexadezimalsystem sind geeignete Zahlensysteme, wenn wir "nah" am Computer arbeiten.

Auflösung des Zahlenrätsels

| | E | D | C | B | A |
|----|---|---|---|---|---|
| 0 | | | | | |
| 1 | | | | | |
| 2 | | | | | |
| 3 | | | | | |
| 4 | | | | | |
| 5 | | | | | |
| 6 | | | | | |
| 7 | | | | | |
| 8 | | | | | |
| 9 | | | | | |
| 10 | | | | | |
| 11 | | | | | |
| 12 | | | | | |
| 13 | | | | | |
| 14 | | | | | |
| 15 | | | | | |
| 16 | | | | | |
| 17 | | | | | |
| 18 | | | | | |
| 19 | | | | | |
| 20 | | | | | |
| 21 | 1 | 0 | 1 | 0 | 1 |
| 22 | | | | | |
| 23 | | | | | |
| 24 | | | | | |
| 25 | | | | | |
| 26 | | | | | |
| 27 | | | | | |
| 28 | | | | | |
| 29 | | | | | |
| 30 | | | | | |
| 31 | | | | | |



Jede Karte steht für einen Stellenwert und zeigt genau die Zahlen, die in ihrer Dualdarstellung an der zugehörigen Stelle eine 1 haben. Die erste Zahl auf jeder Karte hat nur an dieser Stelle eine 1.

Die gesuchte Zahl ergibt sich als Summe der jeweils ersten Zahlen auf den Trefferkarten.

Bits und Bytes

Die kleinste Informationseinheit auf einem Digitalrechner bezeichnen wir als Bit. Ein Bit kann die logischen Werte 0 (Bit gelöscht) und 1 (Bit gesetzt) annehmen.

8 Bits bilden ein Byte:

| | | | | | | | |
|---|----------------|----------------|----------------|--|----------------|----------------|----------------|
| Höchstwertigstes oder most significant Bit | | | | Niederwertigstes oder least significant Bit | | | |
| Byte | | | | | | | |
| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 2 ⁷ | 2 ⁶ | 2 ⁵ | 2 ⁴ | 2 ³ | 2 ² | 2 ¹ | 2 ⁰ |
| Stellenwerte | | | | | | | |

Als Dualzahl interpretiert kann ein Byte Zahlen von 0 – 255 (hex. 00 – ff) darstellen.

Bit 7 bezeichnen wir als das höchstwertige (most significant), Bit 0 als das niederwertigste (least significant) Bit.

Im Sinne der Interpretation als Dualzahl hat das höchstwertige Bit den Stellenwert $2^7=128$, das niederwertigste den Stellenwert $2^0=1$.

Größere Quantitäten

Bei großen Datenmengen orientiert man sich an den Maßeinheitenpräfixen (Kilo, Mega, ...) der Physik.

Anders als in der Physik, in der diese Präfixe immer eine Vervielfachung um den Faktor $10^3=1000$ bedeuten, verwendet man in der Informatik den Faktor $2^{10}=1024$.

| Physik (SI-Präfixe) | | Informatik (Binärpräfixe) | | rel. Abweichung |
|---------------------|-----------|---------------------------|------|-----------------|
| Kilo | 10^3 | 2^{10} | Kibi | 2,40% |
| Mega | 10^6 | 2^{20} | Mebi | 4,86% |
| Giga | 10^9 | 2^{30} | Gibi | 7,37% |
| Tera | 10^{12} | 2^{40} | Tebi | 9,95% |
| Peta | 10^{15} | 2^{50} | Pebi | 12,59% |
| Exa | 10^{18} | 2^{60} | Exbi | 15,29% |

SI = Système international d'unités

Eigentlich sollte man konsequent die Binärpräfixe verwenden. Zumeist verwendet man in der Informatik aber die SI-Präfixe und meint dabei die Werte der Binärpräfixe.

Vorzeichenlose und vorzeichenbehaftete Zahlen

Vorzeichenlose Zahlen werden intern in ihrer Dualdarstellung gespeichert. Bei einer n-Bit Darstellung können auf diese Weise Zahlen von 0 bis 2^n-1 dargestellt werden.

Bei vorzeichenbehaftete Zahlen wird das sogenannte Zweierkomplement (\rightarrow Übungen) verwendet. Bei einer n-Bit Darstellung können auf diese Weise Zahlen von -2^{n-1} bis $2^{n-1}-1$ dargestellt werden.

Üblich sind heute 1-Byte (8 Bit), 2 Byte (16 Bit), 4 Byte (32 Bit) und 8 Byte (64 Bit) Darstellungen mit folgenden Darstellungsbereichen:

| | signed | | unsigned | |
|-----------|-------------|---------------|----------|-------------|
| Größe | min | max | min | max |
| 1 Byte | -128 | 127 | 0 | 255 |
| 2 Bytes | -32768 | 32767 | 0 | 65535 |
| 4 Bytes | -2147483648 | 2147483647 | 0 | 4294967295 |
| 8 Bytes | -9,2234E+18 | 9,2234E+18 | 0 | 1,84467E+19 |
| Allgemein | | | | |
| n Bit | 2^{n-1} | $2^{n-1} - 1$ | 0 | $2^n - 1$ |

Zahlkonstanten in unterschiedlichen Systemen

| Zahlenformat | Präfix |
|--------------|---------------|
| Dezimal | (kein Präfix) |
| Oktal | 0 |
| Hexadezimal | 0x |

```
unsigned int zahl;  
  
zahl = 123456; // Dezimalwert  
  
zahl = 0123456; // Oktalwert  
  
zahl = 0x123abc; // Hexadezimalwert
```

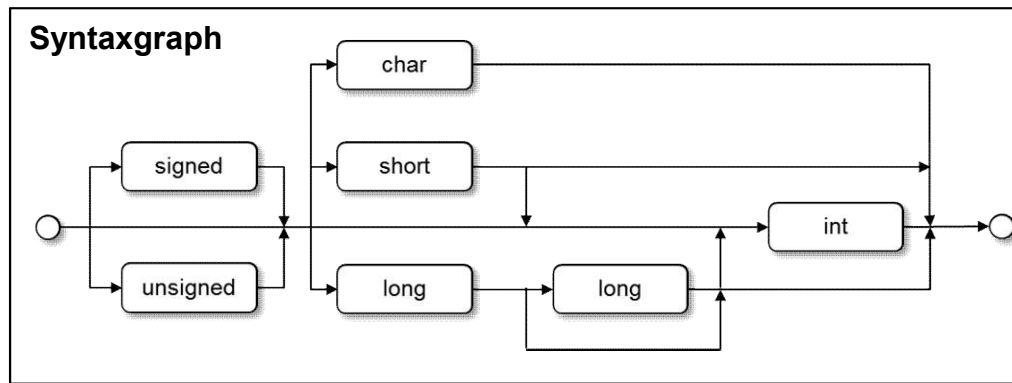
Oktal- und Hexadezimaldarstellung werden in der Regel nur mit vorzeichenlosen Zahlentypen verwendet.

Es gibt in C keine Notation für die Dualdarstellung, man verwendet die Notation für Oktal- bzw. Hexadezimaldarstellung.

Verwenden Sie auf keinen Fall führende Nullen in der Dezimaldarstellung.

Skalare Datentypen in C

Alles, was man beim Durchlaufen des folgenden "Syntaxgraphen" erhalten kann ist ein zulässiger Integer-Datentyp in C:



Beispiele:

```
int a;  
signed char b;  
unsigned short int c;  
long d;  
unsigned long long int e;
```

Im C-Standard ist nur die Mindestgröße für die Datentypen festgelegt. Die tatsächliche Größe ist systemabhängig.

| Datentyp | Mindestgröße | Typische Größe |
|-----------|--------------|----------------|
| char | 1 Byte | 1 Byte |
| short | 2 Bytes | 2 Bytes |
| int | 2 Bytes | 4 Bytes |
| long | 4 Bytes | 4 Bytes |
| long long | 8 Bytes | 8 Bytes |

Darüber hinaus müssen die jeweils kleineren Typen in die jeweils größeren "hineinpassen"

Formatierte Ausgabe in unterschiedlichen Zahlensystemen

Zur Ausgabe in einem bestimmten Zahlensystem werden spezielle Formatanweisungen verwendet:

| Zahlenformat | Ausgabe |
|--------------|---------|
| Dezimal | "%d" |
| Oktal | "%o" |
| Hexadezimal | "%x" |

Beispiele:

```
unsigned int zahl;  
  
zahl = 123456;  
printf( "Dezimalausgabe:      %d\n", zahl);  
printf( "Oktalausgabe:       %o\n", zahl);  
printf( "Hexadezimalausgabe: %x\n", zahl);  
  
zahl = 0123456;  
printf( "Dezimalausgabe:      %d\n", zahl);  
printf( "Oktalausgabe:       %o\n", zahl);  
printf( "Hexadezimalausgabe: %x\n", zahl);  
  
zahl = 0x123abc;  
printf( "Dezimalausgabe:      %d\n", zahl);  
printf( "Oktalausgabe:       %o\n", zahl);  
printf( "Hexadezimalausgabe: %x\n", zahl);
```

```
Dezimalausgabe:      123456  
Oktalausgabe:       361100  
Hexadezimalausgabe: 1e240
```

```
Dezimalausgabe:      42798  
Oktalausgabe:       123456  
Hexadezimalausgabe: a72e
```

```
Dezimalausgabe:      1194684  
Oktalausgabe:       4435274  
Hexadezimalausgabe: 123abc
```

Formatierte Eingabe vorzeichenloser ganzer Zahlen in unterschiedlichen Darstellungen

| Zahlenformat | Eingabe |
|--------------|---------|
| Dezimal | "%d" |
| Oktal | "%o" |
| Hexadezimal | "%x" |

```
unsigned int zahl;  
  
printf( "Dezimaleingabe:      ", &zahl);  
scanf( "%d", &zahl);  
printf( "Dezimalausgabe:      %d\n", zahl);  
printf( "Oktalausgabe:         %o\n", zahl);  
printf( "Hexadezimalausgabe: %x\n\n\n", zahl);  
  
printf( "Oktaleingabe:          ", &zahl);  
scanf( "%o", &zahl);  
printf( "Dezimalausgabe:      %d\n", zahl);  
printf( "Oktalausgabe:         %o\n", zahl);  
printf( "Hexadezimalausgabe: %x\n\n\n", zahl);  
  
printf( "Hexadezimaleingabe: ", &zahl);  
scanf( "%x", &zahl);  
printf( "Dezimalausgabe:      %d\n", zahl);  
printf( "Oktalausgabe:         %o\n", zahl);  
printf( "Hexadezimalausgabe: %x\n\n\n", zahl);
```

```
Dezimaleingabe:      123456  
Dezimalausgabe:      123456  
Oktalausgabe:        361100  
Hexadezimalausgabe:  1e240
```

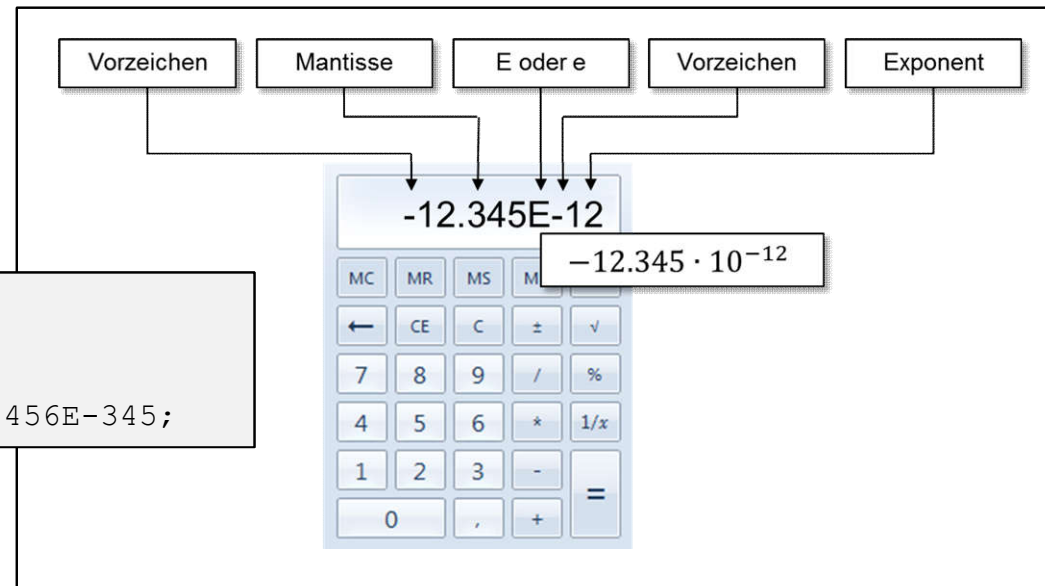
```
Oktaleingabe:        123456  
Dezimalausgabe:      42798  
Oktalausgabe:        123456  
Hexadezimalausgabe:  a72e
```

```
Hexadezimaleingabe:  123abc  
Dezimalausgabe:      1194684  
Oktalausgabe:        4435274  
Hexadezimalausgabe:  123abc
```

Gleitkommazahlen

Gleitkommakonstanten werden im üblichen "Taschenrechnerformat" angegeben.

```
float a = -1;  
float b = 1E2;  
double c = 1.234;  
long double d = -123.456E-345;
```



Bei Ein- bzw. Ausgabe werden die folgenden Formatanweisungen verwendet:

| Typ | Eingabe | Ausgabe |
|-------------|---------|---------|
| float | "%f" | "%f" |
| double | "%lf" | "%lf" |
| long double | "%Lf" | "%LF" |

Beispiel

```
double d;  
  
scanf( "%lf", &d);  
printf( "%lf\n", d);
```

Bitoperationen

Bitoperationen ermöglichen es, auf einzelne oder mehrere Bits einer üblicherweise vorzeichenlosen Zahl gezielt zuzugreifen:

| Bitweises Komplement | | | | | | | | |
|----------------------|---|---|---|---|---|---|---|---|
| x | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| ~x | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |

| Bitweises Und | | | | | | | | |
|---------------|---|---|---|---|---|---|---|---|
| x | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| y | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| x&y | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

| Bitweises Oder | | | | | | | | |
|----------------|---|---|---|---|---|---|---|---|
| x | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| y | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| x y | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |

| Bitshift links | | | | | | | | |
|----------------|---|---|---|---|---|---|---|---|
| x | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| x<<2 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 |

| Bitweises Entweder-Oder | | | | | | | | |
|-------------------------|---|---|---|---|---|---|---|---|
| x | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| y | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| x^y | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |

| Bitshift rechts | | | | | | | | |
|-----------------|---|---|---|---|---|---|---|---|
| x | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 1 |
| x>>2 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 |

Manipulation einzelner Bits in einem Bitmuster

Mit Bitoperationen können einzelne Bits gesetzt, gelöscht oder invertiert werden

Setzen des 3-ten Bits in x

| | | | | | | | | |
|-----------------|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $1 \ll 3$ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| x | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $x (1 \ll 3)$ | 1 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |

Eine 1 wird um 3 Positionen nach links geschoben und über ein bitweises Oder mit x verknüpft. Dadurch wird das dritte Bit in x gesetzt.

Eine 1 wird um 4 Positionen nach links geschoben, komplementiert und dann über ein bitweises Und mit x verknüpft. Dadurch wird das vierte Bit in x gelöscht.

Löschen des 4-ten Bits in x

| | | | | | | | | |
|-----------------------|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $1 \ll 4$ | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| $\sim (1 \ll 4)$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| x | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $x \& \sim (1 \ll 4)$ | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |

Eine 1 wird um 5 Positionen nach links geschoben und über ein bitweises Entweder-Oder mit x verknüpft. Dadurch wird das fünfte Bit in x invertiert.

Invertieren des 5-ten Bits in x

| | | | | | | | | |
|----------------------|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $1 \ll 5$ | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| x | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| $x \wedge (1 \ll 5)$ | 1 | 0 | 0 | 1 | 0 | 0 | 1 | 1 |

Typische Verwendung von Bitoperationen

Setzen, Löschen, Invertieren von Bits

```
int n = 3;
unsigned int x = 0xaffe;

x = x | (1<<n); // Setzen des n-ten Bits in x
x = x & ~(1<<n); // Löschen des n-ten Bits in x
x = x ^ (1<<n); // Invertieren des n-ten Bits in x
```

Testen von Bits

```
int n = 3;
unsigned int x = 0xaffe;

if( x & (1<<n)) // Test, ob das n-te Bit in x gesetzt ist
{
    ...
}
```

Programmierung des Kartentricks

```
void main()
{
    int bit, z, zahl, antwort;

    printf( "Denk dir eine Zahl zwischen 0 und 31\n");

    for( bit = 1, zahl = 0; bit < 32; bit = bit << 1)
    {
        printf( "Ist die Zahl in dieser Liste");
        for( z = 0; z < 32; z++)
        {
            if( z & bit)
                printf( " %d", z);
        }
        printf( ":" );
        scanf( "%d", &antwort);
        if( antwort == 1)
            zahl = zahl | bit;

        printf( "Die Zahl ist %d\n", zahl);
    }
}
```

Die Variable `bit` durchläuft in der Schleife die Werte

1 = 00001₂
2 = 00010₂
4 = 00100₂
8 = 01000₂
16 = 10000₂

Hier werden alle 32 Zahlen durchlaufen, aber ausgegeben werden nur die, die das `bit` gesetzt haben

Wenn die Zahl auf der Karte steht, wird das `bit` gesetzt.

```
Denk dir eine Zahl zwischen 0 und 31
Ist die Zahl in dieser Liste 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31: 1
Ist die Zahl in dieser Liste 2 3 6 7 10 11 14 15 18 19 22 23 26 27 30 31: 0
Ist die Zahl in dieser Liste 4 5 6 7 12 13 14 15 20 21 22 23 28 29 30 31: 1
Ist die Zahl in dieser Liste 8 9 10 11 12 13 14 15 24 25 26 27 28 29 30 31: 0
Ist die Zahl in dieser Liste 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31: 1

Die Zahl ist 21
```

Zahlenraten

```
void main()
{
    int n, antwort;
    unsigned int zahl, bit;

    printf( "Anzahl Stellen: ");
    scanf( "%d", &n);
    printf( "Denk dir eine Zahl zwischen 0 und %d\n", (1<<n)-1);

    for( zahl = 0, bit = (1<<n-1); bit > 0; bit = (bit>>1))
    {
        zahl = zahl | bit;
        printf( "Ist die Zahl kleiner als %d: ", zahl);
        scanf( "%d", &antwort);
        if( antwort == 1)
            zahl = zahl & ~bit;
    }

    printf( "Die Zahl ist %d\n", zahl);
}
```

Es werden n-stellige Dualzahlen betrachtet. Die größte n-stellige Dualzahl ist:

$$(1 \ll n) \quad 1 = 2^n \quad 1 = \underbrace{111 \dots 1}_{n \text{ mal}}$$

Die Variable `bit` durchläuft Potenzen von 2:

$$2^{n-1} = 1000 \dots 000_2$$

$$2^{n-2} = 0100 \dots 000_2$$

$$2^{n-3} = 0010 \dots 000_2$$

...

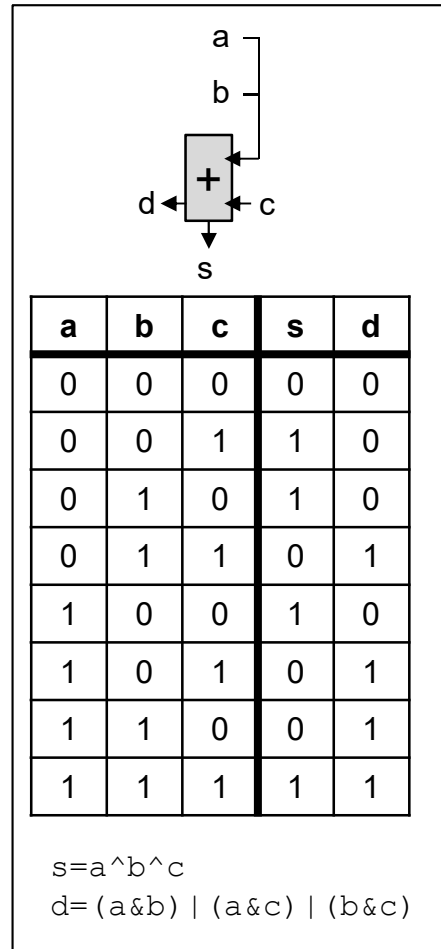
$$2^1 = 0000 \dots 010_2$$

$$2^0 = 0000 \dots 001_2$$

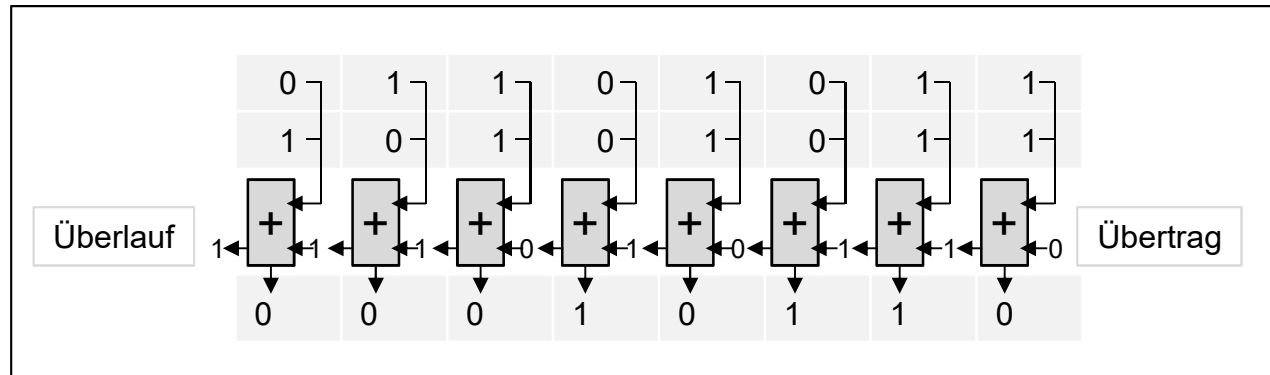
Das Bit wird zunächst gesetzt. Wenn die Zahl dann zu groß ist, wird das Bit wieder gelöscht.

```
Anzahl Stellen: 16
Denk dir eine Zahl zwischen 0 und 65535
Ist die Zahl kleiner als 32768: 1
Ist die Zahl kleiner als 16384: 1
Ist die Zahl kleiner als 8192: 1
Ist die Zahl kleiner als 4096: 0
Ist die Zahl kleiner als 6144: 1
Ist die Zahl kleiner als 5120: 1
Ist die Zahl kleiner als 4608: 0
Ist die Zahl kleiner als 4864: 1
Ist die Zahl kleiner als 4736: 1
Ist die Zahl kleiner als 4672: 0
Ist die Zahl kleiner als 4704: 0
Ist die Zahl kleiner als 4720: 1
Ist die Zahl kleiner als 4712: 1
Ist die Zahl kleiner als 4708: 0
Ist die Zahl kleiner als 4710: 0
Ist die Zahl kleiner als 4711: 0
Die Zahl ist 4711
```


Addierwerk



Die Bits werden aus den Zahlen gefiltert
danach werden Summe und Übertrag für
die betrachtete Stelle berechnet.



```

void main()
{
    unsigned int z1, z2;
    unsigned int s, s1, s2, sum, c;

    printf( "Gib bitte zwei Zahlen ein: ");
    scanf( "%d %d", &z1, &z2);

    for( sum = 0, s = 1, c = 0; s != 0; s = s << 1, c = c << 1)
    {
        s1 = z1 & s;
        s2 = z2 & s;
        sum = sum | (s1 ^ s2 ^ c);
        c = (s1 & s2) | (s1 & c) | (s2 & c);
    }

    printf( "Summe: %d\n", sum);
}

```

z1, z2 sind die zu addierenden Zahlen
 s ist die Maske, die über die Zahlen
geschoben wird.
 s1, s2 sind die aus den Zahlen z1 und z2
maskierten Bits.
 sum ist die zu berechnende Summe
 c ist der Übertrag (carry).

Maske und Carry werden über die
Zahlen geschoben

```

Gib bitte zwei Zahlen ein: 12345 67890
Summe: 80235

```

Zeichen

Zeichen oder Buchstaben werden als "kleine" vorzeichenlose Zahlen (`unsigned char`) gespeichert.

Jedes Zeichen hat seinen speziellen Code:

ASCII-Zeichentabelle, **hexadezimale** Nummerierung

| Code | ...0 | ...1 | ...2 | ...3 | ...4 | ...5 | ...6 | ...7 | ...8 | ...9 | ...A | ...B | ...C | ...D | ...E | ...F |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 0... | NUL | SOH | STX | ETX | EOT | ENQ | ACK | BEL | BS | HT | LF | VT | FF | CR | SO | SI |
| 1... | DLE | DC1 | DC2 | DC3 | DC4 | NAK | SYN | ETB | CAN | EM | SUB | ESC | FS | GS | RS | US |
| 2... | SP | ! | " | # | \$ | % | & | ' | (|) | * | + | , | - | . | / |
| 3... | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | : | ; | < | = | > | ? |
| 4... | @ | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O |
| 5... | P | Q | R | S | T | U | V | W | X | Y | Z | [| \ |] | ^ | _ |
| 6... | ` | a | b | c | d | e | f | g | h | i | j | k | l | m | n | o |
| 7... | p | q | r | s | t | u | v | w | x | y | z | { | | } | ~ | DEL |

Quelle: Wikipedia

Das Zeichen Z hat den ASCII-Code $5a_{16} = 90_{10}$. Der numerische Wert ist dabei relativ unwichtig. Wichtig ist das Bitmuster:

5a = 0101 1010

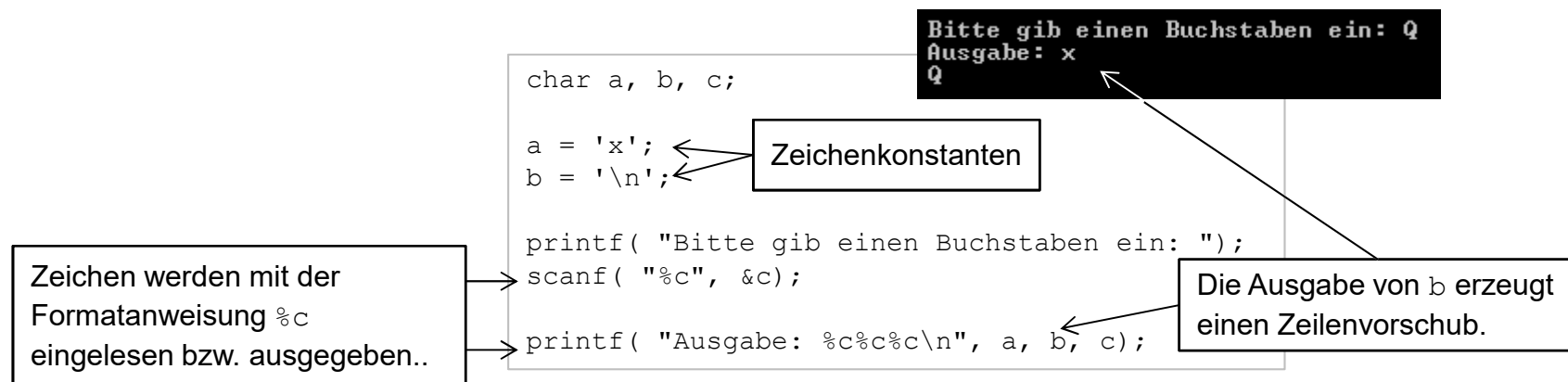
Der ASCII-Zeichensatz enthält keine Codes für speziellen Zeichen wie Umlaute oder ß.

Ein- und Ausgabe von Zeichen

Zeichenkonstanten werden in einfache Hochkommata gesetzt ('a' ... 'z'). Für das nicht druckbare Zeilenvorschub-Zeichen verwenden wir die Ersatzdarstellung '\n'.

Als Typ für Zeichenvariablen wird `char` (oder `unsigned char`) verwendet.

Bei der Ein- beziehungsweise Ausgabe einzelner Zeichen wird "%c" als Formatanweisung verwendet:



Beachten Sie, dass beim Lesen mit %c nur das eine Zeichen eingelesen wird und zusätzlich eingegebene Zeichen, wie das unvermeidliche Linefeed zum Abschluss der Eingabe, im Eingabepuffer verbleiben und dann ggf. bei der nächsten Leseoperation gelesen werden.

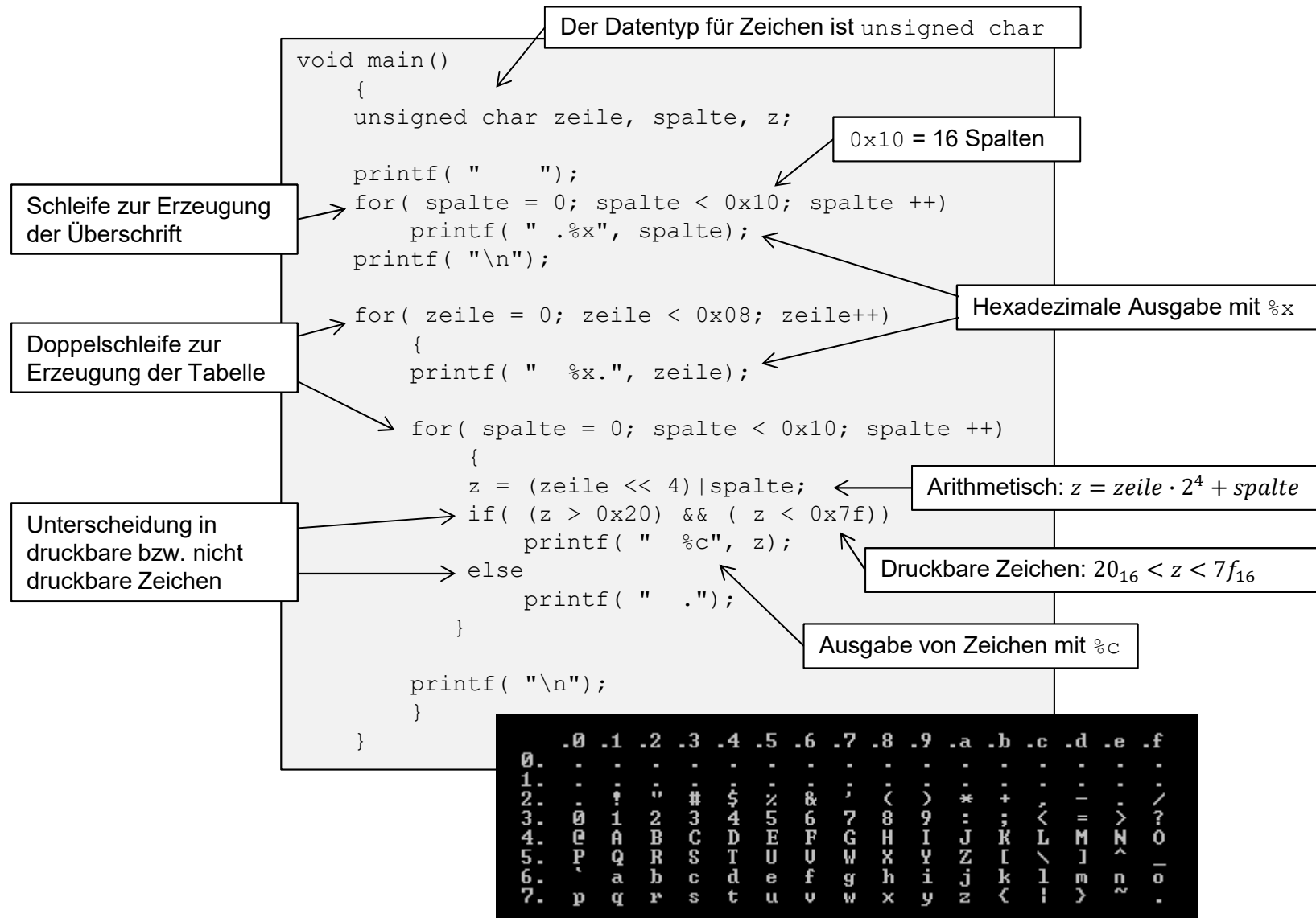
Intern sind Zeichen (kleine) Zahlen

Dass Zeichen (`char`) Zahlen sind, hat den positiven Seiteneffekt, dass mit Zeichen, wie mit Zahlen, gerechnet werden kann:

```
char x;  
  
for( x = 'A'; x <= 'K'; x = x + 1)  
    printf( "%d %c\n", x, x);
```

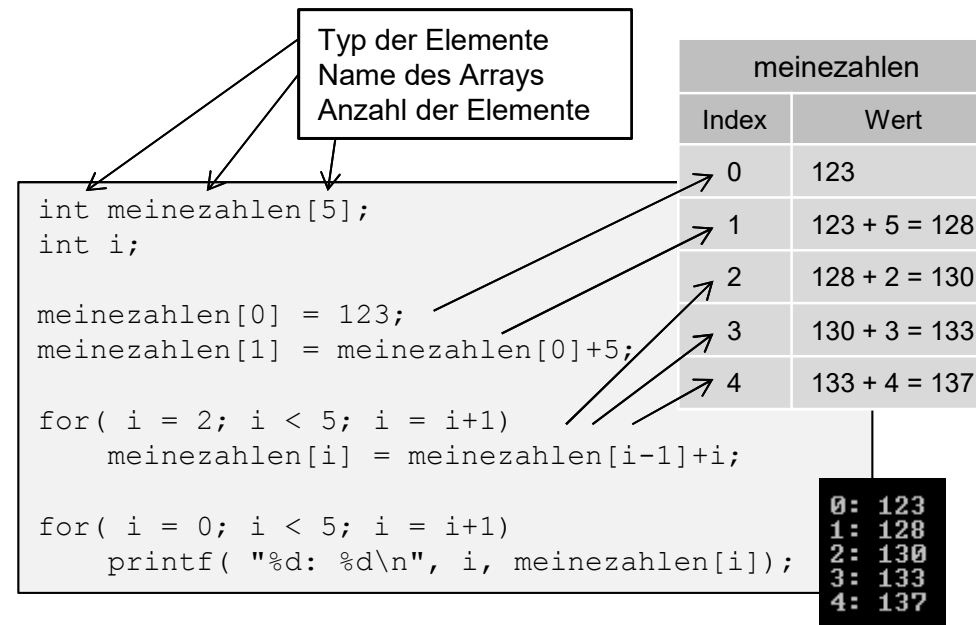
```
65: A  
66: B  
67: C  
68: D  
69: E  
70: F  
71: G  
72: H  
73: I  
74: J  
75: K
```

Da die Nummerierung der Zeichen im ASCII-Code der alphabetischen Reihenfolge entspricht, kann dies genutzt werden, um Zeichen alphabetisch zu sortieren.



Arrays

Ein **Array** ist eine Aneinanderreihung von Datenelementen gleichen Typs. Über einen Index kann auf jedes Datenelement unmittelbar zugegriffen werden.



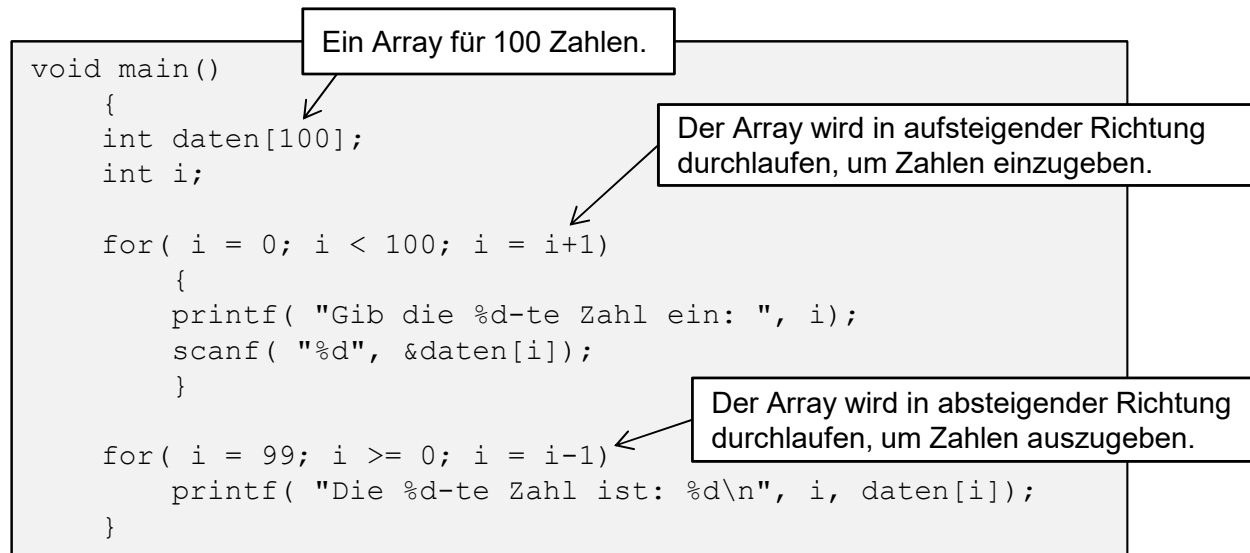
Arrays können direkt bei ihrer Definition mit Werten gefüllt werden:

```
int meinezahlen[5] = { 1, 2, 3, 4, 5};
```

Achtung: Die Nummerierung der Elemente im Array beginnt mit dem Index 0. Hat ein Array 5 Elemente, so sind diese von 0 bis 4 nummeriert. Der Programmierer muss darauf achten, nur gültige Indices zu verwenden. Bei der Verwendung ungültiger Indices stürzt das Programm in der Regel ab.

Programmierbeispiel

Lies 100 Zahlen von der Tastatur ein und gib sie in umgekehrter Reihenfolge wieder aus.



Zweidimensionale Arrays

Entfernungstabelle

Eine **Entfernungstabelle** gibt die Entfernung zwischen zwei geographischen Orten an, wobei es sich häufig um Großstädte handelt. Die Entfernung wird dabei in der **Luftlinie** oder für ein bestimmtes Verkehrsmittel angegeben.

Beispiel

| | A | B | C | D | E |
|---|----|----|----|----|----|
| A | . | 2 | 5 | 9 | 14 |
| B | 2 | . | 7 | 15 | 27 |
| C | 5 | 7 | . | 9 | 23 |
| D | 9 | 15 | 9 | . | 12 |
| E | 14 | 27 | 23 | 12 | . |

```
void main()
{
    int start, ziel, distanz;
    int entfernung[5][5] = {
        { 0, 2, 5, 9, 14},
        { 2, 0, 7, 15, 27},
        { 5, 7, 0, 9, 23},
        { 9, 15, 9, 0, 12},
        { 14, 27, 23, 12, 0}
    };
}
```

Zeilen- und Spaltenindex
werden eingelesen.

Hier wird ein zweidimensionaler
Array für 5x5 int-Werte
angelegt und initialisiert.

```
printf( "Gib zwei Orte (0-4) ein: ");
scanf( "%d %d", &start, &ziel);
```

```
distanz = entfernung[start][ziel];
```

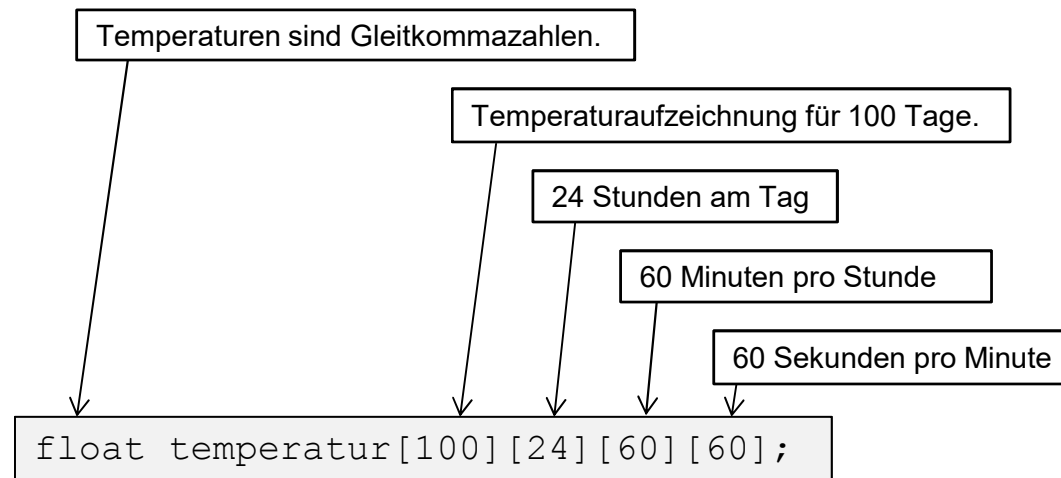
```
printf( "Entfernung zwischen %d und %d: %d km\n", start, ziel, distanz);
}
```

Die Entfernung wird aus der Tabelle
gelesen und ausgegeben.

```
Gib zwei Orte (0-4) ein: 0 3
Entfernung zwischen 0 und 3: 9 km
```


Arrays mit mehr als zwei Dimensionen

Wenn man über einen Zeitraum von 100 Tagen sekundlich die Temperatur aufzeichnen will, kann man einen vierdimensionalen Array verwenden:



Die Temperatur am 5 Tag der Aufzeichnungen um 10 Sekunden nach 14:00 Uhr erhält man dann mit dem Zugriff:

```
t = temperatur[4][14][0][10]
```

Beachten Sie, dass die Zählung der Tage, Stunden, Minuten und Sekunden im Array mit 0 beginnt.

Zeichenketten

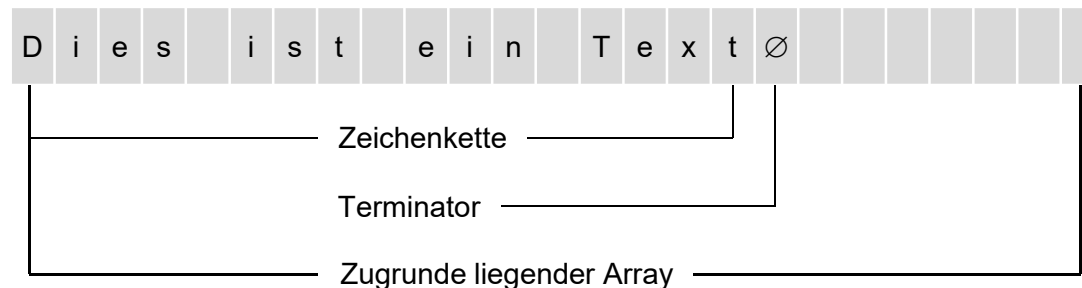
Konstante Zeichenketten werden in doppelte Hochkommata eingeschlossen:

```
"Hello World\n"
```

Beachten Sie den Unterschied:

- 'A' ist der Buchstabe A
- "A" ist eine Zeichenkette, die nur den Buchstaben A enthält

Eine Zeichenkette wird in einem Array mit Basistyp `char` (oder `unsigned char`) gespeichert und durch einen Terminator (0) abgeschlossen:

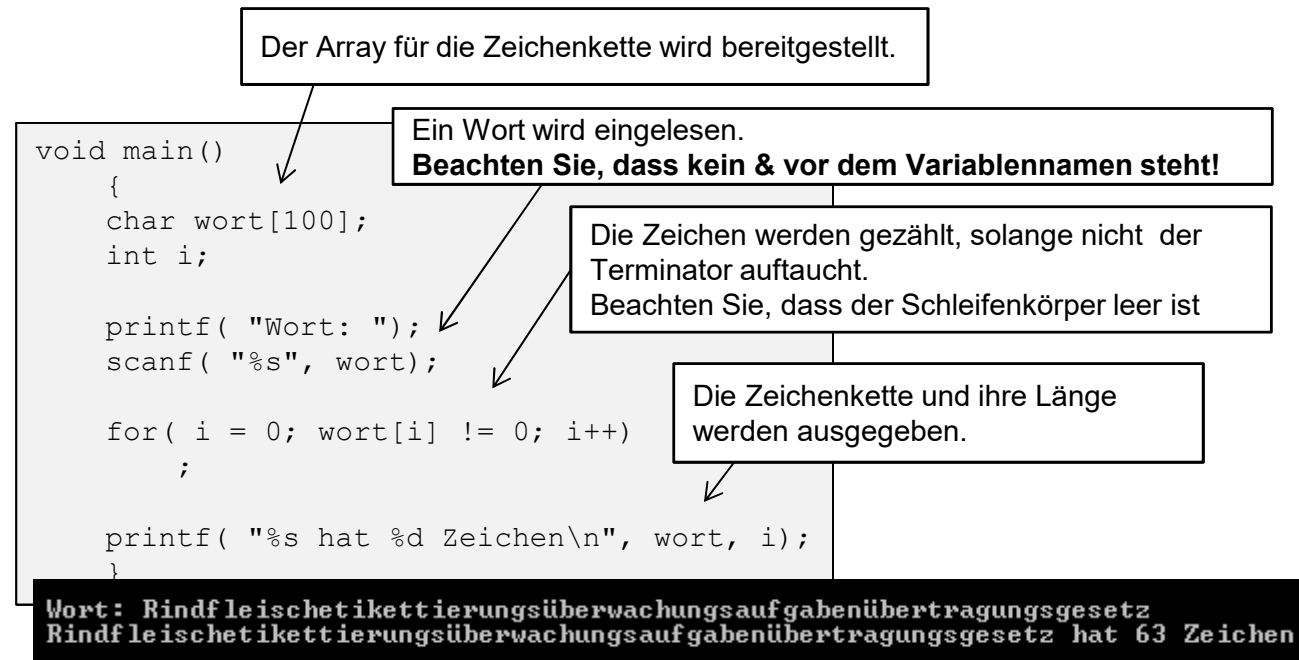


Der Array kann größer als benötigt sein, er muss aber, wegen des Terminators, mindestens $n+1$ Elemente haben, um eine Zeichenkette der Länge n aufzunehmen.

Bei dem Terminator handelt es sich um die (binäre) 0 nicht um das Zeichen '0'.

Ein- und Ausgabe von Zeichenketten

Zeichenketten werden mit der Formatanweisung `%s` in einen Array eingelesen und mit der Formatanweisung `%s` ausgegeben.



Der Terminator wird beim Einlesen automatisch hinzugefügt. Am Terminator erkennt ein Programm das Wortende.

Beim Einlesen von Zeichenketten mit `scanf` wird dem Variablennamen kein `&` vorangestellt . Wenn Sie ein `&` setzen, wird Ihr Programm abstürzen.

Bei der Eingabe wird nicht geprüft, ob der Array groß genug ist, um den String aufzunehmen. Werden mehr Zeichen eingegeben, als der Array aufnehmen kann, stürzt das Programm ab.

Manipulation von Zeichenketten

Zeichenketten können durch Zugriff in den Array nach Belieben erzeugt oder verändert werden:

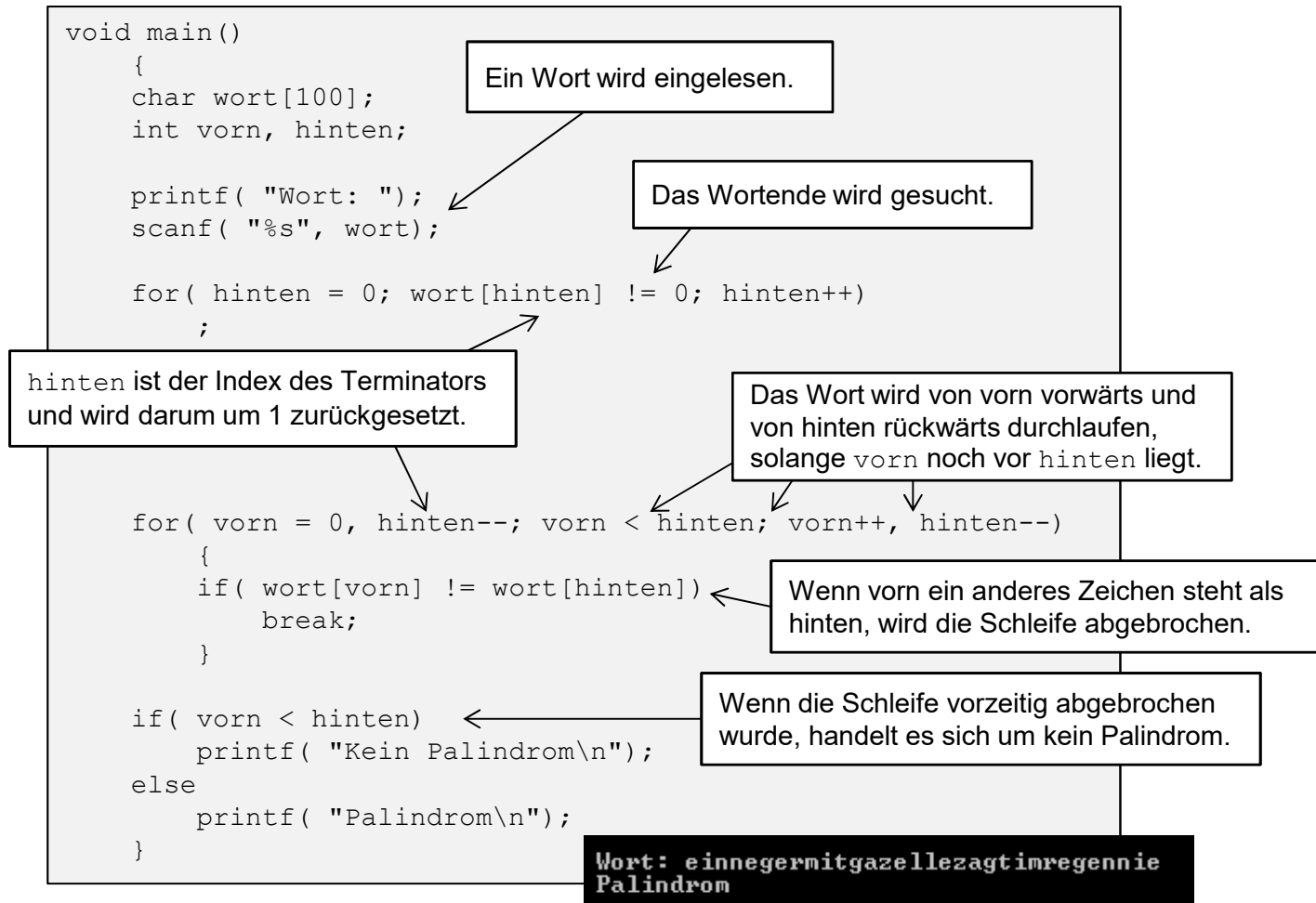
```
char wort[20];  
  
wort[0] = 'A';      // Erster Buchstabe A  
wort[1] = wort[0];  // Zweiter Buchstabe auch A  
wort[2] = 0;        // Terminator, Zeichenkette ist "AA"
```

Dabei ist folgendes unbedingt zu beachten:

- Die Nummerierung der Zeichen beginnt beim Index 0. Wenn die Zeichenkette n Zeichen hat, sind diese von 0 bis n-1 nummeriert. Der Terminator hat den Index n.
- Die Zeichenkette befindet sich in einem Array fester Länge. Sie müssen darauf achten, dass bei Veränderungen (zum Beispiel durch Anfügen von Buchstaben) die Grenzen des zugrunde liegenden Arrays nicht überschritten werden.
- Wegen des Terminators muss der Array, der den String aufnimmt, mindestens ein Element mehr haben als der String Zeichen enthält.
- Die Zeichenkette muss nach Manipulationen immer konsistent sein. Insbesondere bedeutet das, dass das Terminator-Zeichen korrekt positioniert werden muss.

Auf die Rahmenbedingungen hat der Programmierer zu achten. Verletzt er eine dieser Bedingungen, stürzt das Programm in der Regel ab.

Palindromerkennung



Galgenmännchen

```
void main()
{
    char wort[100], anzeige[100];
    int versuch;
    int nochzuraten, i, anzahl;

    printf( "Wort: ");
    scanf( "%s", wort);

    for( nochzuraten = 0; wort[nochzuraten] != 0; nochzuraten++)
        anzeige[nochzuraten] = '-';
    anzeige[nochzuraten] = 0;

    for( anzahl = 1; nochzuraten != 0; anzahl++)
    {
        printf( "%s\n", anzeige);
        printf( "%d-ter Versuch: ", anzahl);

        scanf( "\n%c", &versuch);

        for( i = 0; wort[i] != 0; i++)
        {
            if( (wort[i] == versuch) && (anzeige[i] == '-'))
            {
                anzeige[i] = versuch;
                nochzuraten--;
            }
        }

        printf( "%s\n", anzeige);
    }
    printf( "Du hast %d Versuche benoetigt\n", anzahl-1);
}
```

Puffer für das Ratewort und
den Anzeigestring.

Das zu ratende Wort wird eingelesen.

Der Anzeigestring wird aufbereitet, indem für
alle Zeichen ein '-' gesetzt wird.
Gleichzeitig wird gezählt, wie viele Zeichen
zu raten sind.

Der Anzeigestring wird terminiert.

Schleife über alle Rateversuche.

Eingabe eines neuen Zeichens.
Vor dem Lesen wird mit \n der noch in
der Eingabe stehende Zeilenvorschub
aus der letzten Eingabe konsumiert.

Schleife über das zu ratende Wort.

Wenn der geratene Buchstabe mit dem Zeichen
im Wort übereinstimmt und in der Anzeige noch
ein '-' steht, wird das Zeichen in die Anzeige
übernommen und es ist nur noch ein Buchstabe
weniger zu erraten.

Wort: mississippi

```
1-ter Versuch: i
-i--i--i--i
2-ter Versuch: a
-i--i--i--i
3-ter Versuch: s
-issisi--i
4-ter Versuch: p
-ississippi
5-ter Versuch: m
mississippi
Du hast 5 Versuche benoetigt
```

Zeichenketten können nicht mit = kopiert werden

Zum Erstellen einer Kopie, muss die Zeichenkette Zeichen für Zeichen kopiert werden.

Die Kopie muss mit dem Terminatorzeichen abgeschlossen werden.

```
char original[100], kopie[100];
int i;

printf( "Eingabe:  ");
scanf( "%s", original);

for( i = 0; original[i] != 0; i++)
    kopie[i] = original[i];
kopie[i] = 0;

printf( "\nOriginal: %s", original);
printf( "\nKopie:      %s", kopie);
```

Zeichen für Zeichen wird von original nach kopie kopiert.

Hier wird die Kopie terminiert.

Eingabe: Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz
Original: Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz
Kopie: Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz

Später werden wir eine "Funktion" zum Kopieren von Zeichenketten kennenlernen.

Zeichenketten können nicht mit == verglichen werden

Zum Vergleich müssen die Zeichenketten Zeichen für Zeichen miteinander verglichen werden.

```
char wort1[100], wort2[100];  
int i;
```

```
printf( "Wort1:  ");  
scanf( "%s", wort1);  
printf( "Wort2:  ");  
scanf( "%s", wort2);
```

Zeichen für Zeichen wird geprüft.
Solange das erste Wort noch nicht beendet
ist und die Zeichen im ersten und zweiten
Wort gleich sind, wird weiter geprüft.

```
for( i = 0; (wort1[i] != 0) && (wort1[i] == wort2[i]); i++)  
    ;
```

```
if( wort1[i] == wort2[i])  
    printf( "Die Worte sind gleich\n");  
else  
    printf( "Die Worte sind verschieden");
```

Am zuletzt geprüften Zeichen
kann man erkennen, ob die
beiden Worte gleich waren.

```
Wort1: Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz  
Wort2: Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz  
Die Worte sind verschieden
```

Später werden wir eine "Funktion" zum Vergleichen von Zeichenketten kennenlernen.

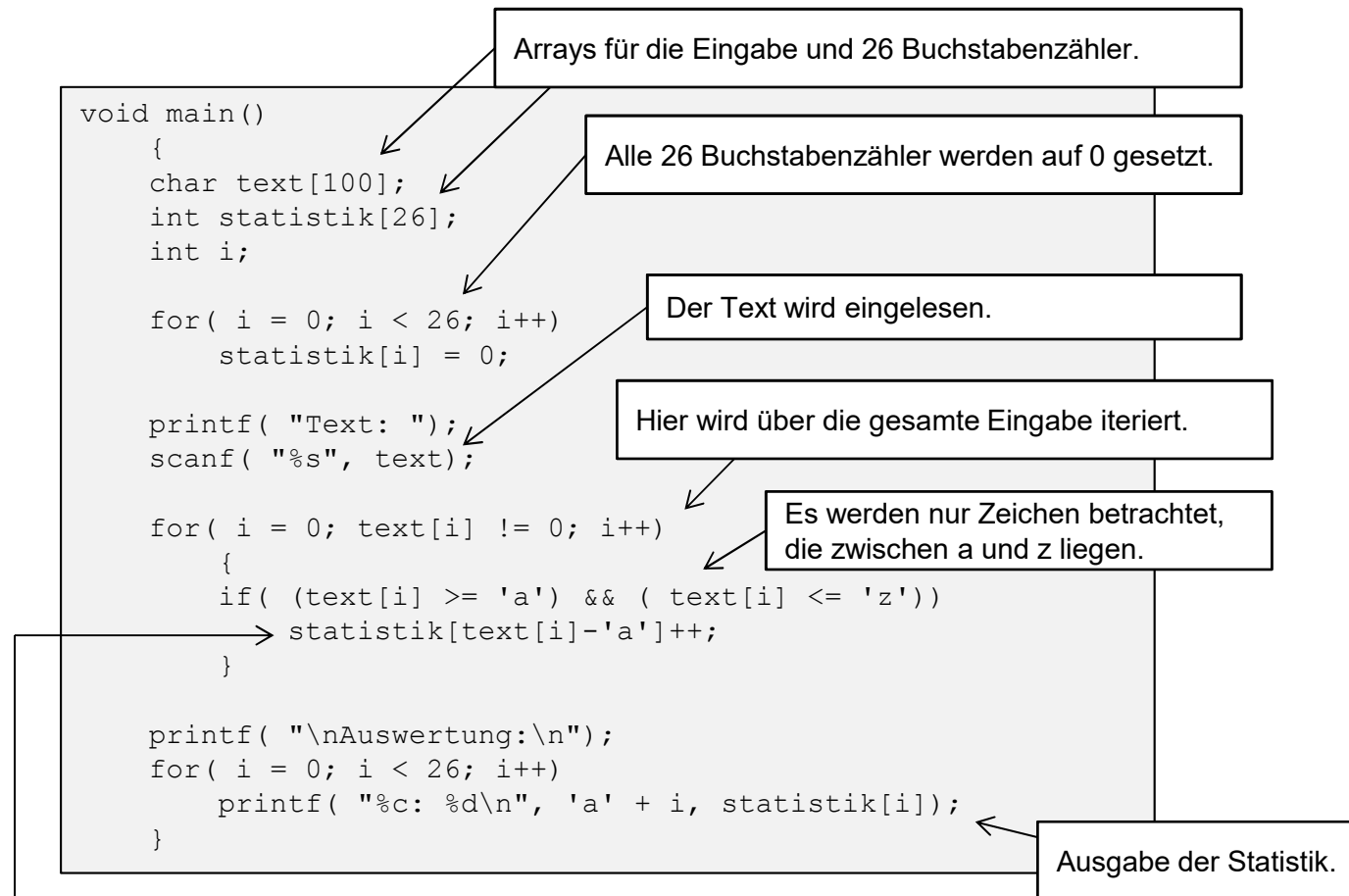
Beispiel

Erstelle ein Programm, das eine Textzeile von der Tastatur einliest und eine Statistik über das Vorkommen von Buchstaben berechnet.

So soll das Programm ablaufen:

```
Text: franzjagtinkomplettverwahrlostentaxiquerdurchbayern
Auswertung:
a: 5
b: 1
c: 1
d: 1
e: 5
f: 1
g: 1
h: 2
i: 2
j: 1
k: 1
l: 2
m: 2
n: 3
o: 2
p: 1
q: 1
r: 6
s: 1
t: 5
u: 2
v: 1
w: 1
x: 1
y: 1
z: 1
```

Buchstabenstatistik



Das Zeichen wird im Array `statistik` gezählt:

| | |
|---|---|
| <code>text[i]</code> | ist das i-te Zeichen im Text |
| <code>text[i] - 'a'</code> | ist der Index des Zählers für das Zeichen <code>text[i]</code> ('a' → 0, 'b' → 1, ... 'z' → 25) |
| <code>statistik[text[i] - 'a']</code> | ist der Statistikzähler für das Zeichen <code>text[i]</code> |
| <code>statistik[text[i] - 'a']++</code> | bedeutet, dass der Statistikzähler für das Zeichen <code>text[i]</code> um 1 erhöht wird |