Zeichen

Zeichen werden als ganze Zahlen im Bereich 0-127 repräsentiert (die ersten 32 Zeichen sind Steuerzeichen, die folgenden 96 sichtbare). Um gegenüber `int` Speicherplatz zu sparen, existiert ein Ganzzahltyp `char`, welcher je nach Plattform Zahlen von -128 bis +127 (entspricht `signed char`) oder 0 bis 255 (entspricht `unsigned char`) darstellen kann.

Der Compiler unterstützt den Programmierer durch die Bereitstellung von Literalen, die in einfachen Hochkommata angegeben werden – diese generieren zur Übersetzungszeit die entsprechenden Zahlen (im ASCII-Code wird aus `'a'` z.B. 97, aus `'0'` wird 48 usw).

Die Umwandlung zwischen Zahlen und Zeichen geschieht nur während der Ein- und Ausgabe. `char` speichert Zahlen, mit denen ganz normal gerechnet werden kann:

```
printf("%d + %d = %d\n", 'a', '0', 'a' + '0');
```

97 + 48 = 145

Um Zeichen auf die Konsole zu schreiben, verwenden man den Format-String `%c`:

```
printf("%c + %d = %c\n", 'a', 1, 'a' + 1);
```

a + 1 = b

String-Literale

String-Literale werden im Quelltext durch Anführungszeichen begrenzt und enden immer implizit mit dem besonderen, unsichtbaren Zeichen `'\0'`, welches den ASCII-Wert 0 hat (nicht zu verwechseln mit dem Zeichen `'0'`, welches den ASCII-Wert 48 hat). Dieser sog. NUL-Terminator markiert das Ende eines String-Literals und zählt zur Größe dazu:

```
printf("%u\n", sizeof "Hallo");
```

6

Die Anzahl Zeichen vor der terminierenden NUL liefert die Bibliotheksfunktion `strlen`:

```
#include <string.h>
…
printf("%u\n", strlen("Hallo"));
```

5

Der Zeilenumbruch `\n` ist ein Beispiel für ein besonderes Zeichen, welches im Quelltext durch mehrere Zeichen symbolisiert wird. Weitere Beispiele sind `\t` und `\"` und `\\`.

```
printf("%u\n", strlen("\n\t\"\\"));
```

4

Direkt aufeinanderfolgende String-Literale werden zu einem String verschmolzen:

```
printf("%s\n", "Dampfschiff" "fahrt" "s" "kapitän")
```

Dampfschifffahrtskapitän

String-Literale werden üblicherweise in schreibgeschützten Speicherbereichen abgelegt:

```
"Hallo."[5] = '!';   // unzulässig, stürzt wahrscheinlich ab
```

String-Variablen

Für veränderliche Strings verwendet man Variablen, die Arrays von Zeichen sind:

```
char a[] = "Hallo.";
a[5] = '!';
printf("%s\n", a);
```

Hallo!

Das Array `a` hat die Größe 7, damit die 6 sichtbaren Zeichen und die terminierende NUL darin Platz finden. Mehr Speicherplatz ist nicht vorhanden!

```
a[6] = '?';          // überschreibt die terminierende NUL
printf("%s\n", a);   // keine gute Idee...
```

Hallo!?<zufälliger Müll>

Die Funktion printf wird nach `Hallo!?` noch so lange weitere Zeichen aus dem Speicher auf die Konsole schreiben, bis es zufällig auf eine terminierende NUL trifft. Das ist recht wahrscheinlich, so dass solche Programmierfehler häufig längere Zeit unentdeckt bleiben.

Wenn man weiß, dass man später noch weitere Zeichen in dem Array ablegen will, dann muss man die gewünschte Maximallänge schon bei der Definition explizit angeben:

```
char b[50] = "Hallo!";
printf("%u %u\n", sizeof b, strlen(b));
```

50 6

```
b[6] = '?';
printf("%u %u\n", sizeof b, strlen(b));
```

50 7

In vielen Sprachen kann man Strings mit den beiden Operatoren = und + kopieren und konkatentieren. In C verwendet man stattdessen die Funktionen `strcpy` und `strcat`:

```
char c[50];
strcpy(c, "Hallo ");
strcat(c, "Welt!");
```

Genau wie bei der Übergabe von anderen Arrays lässt man bei der Übergabe von Strings die Länge in den eckigen Klammern weg. Bei bereits initialisierten Strings muss man die Länge nicht explizit übergeben, sondern kann sie mit der Funktion `strlen` ermitteln:

```
void lese_aus_string(char s[])
{
    int len = strlen(s); …
}
```

Auch hier wird intern nur die Information übergeben, wo der String im Speicher anfängt.

Zum Vergleichen von Strings dient die Funktion `int strcmp(char a[], char b[])`. Wenn a in einem Wörterbuch vor b stünde, liefert die Funktion irgendeine negative Zahl. Wenn a in einem Wörterbuch nach b stünde, liefert die Funktion irgendeine positive Zahl. Wenn a und b dasselbe Wort repräsentieren, liefert die Funktion die Zahl 0.