10. String und Stringverarbeitung

Zeichenketten sind eindimenionale Felder mit mehreren Besonderheiten. Sie können auch in der Speicherklasse `auto` initialisiert werden. Als Zeichenkette müssen sie mit `'\0'` abgeschlossen sein. Dies lässt sich aber auch automatisch mit Hilfe der vereinfachten Initialisierung erreichen:

```
char s[] = {'s','t','r','i','n','g','\0'};
char s[] = "string";       /* \0 intern angehaengt */
char s[10] = "string";     /* restl. Elemente mit 0 init. */
```

Im Übrigen werden Zeichenketten wie normale Vektoren behandelt. Insbesondere ist der Zugriff auf Vektorelemente gleich:
`s[0] hat den Wert 's'`
`s[3] hat den Wert 'i'`

Wichtig: Die Zeichen*ketten*konstante "x" unterscheidet sich signifikant von der Zeichenkonstante 'x'. Letztere ist ein einzelnes Zeichen, das man z. B. durch `char ch = 'x';` definieren könnte. Bei "x" handelt es sich um ein Zeichenkettenarray, das aus zwei Zeichen besteht ('x' und '\0').

char-Arrays können auch mit einer String-Konstanten initialisiert werden. Statt

```
char s[] = {'s','t','r','i','n','g','\0'};
```

kann man auch schreiben

```
char s[] = "string";
```

In C gibt es keine speziellen Sprachelemente für die Manipulation von Zeichenketten. Es existieren jedoch in der C-Standardbibliothek eine Reihe von Funktionen zur Stringbearbeitung.

- Funktionen zum Kopieren von Strings und Zeichenarrays
- Funktionen zum Aneinanderhängen von Strings
- Funktionen zum Vergleich von Strings und Zeichenarrays
- Funktionen zum Suchen nach Zeichen und Zeichenfolgen in Strings und Zeichenarrays
- Funktion zur Ermittlung der Länge eines Strings
- Funktion zum Füllen eines Arrays mit einem Zeichen
- Funktion zur Ermittlung der zu einer Fehlernummer gehörenden Fehlermeldung

Die entsprechenden Funktions-Deklarationen befinden sich in der Standard-Header-Datei <string.h>. Diese ist daher bei Verwendung der Funktionen mittels `#include <string.h>` einzubinden. In `<string.h>` ist auch der von einigen Funktionen verwendete Datentyp `size_t` definiert. Dies ist der "natürliche" unsigned-Typ, d. h. der Typ, den der sizeof-Operator liefert.

Da in C Array-Grenzen-Überschreitungen nicht überprüft werden, liegt es am Programmierer einen Array-Überlauf zu verhindern. Im ANSI-Standard ist lediglich festgelegt, daß das Überschreiten von Array-Grenzen zu einem undefinierten Verhalten führt. Alle

Kopierfunktionen (siehe unten) setzen voraus, daß sich der Quell- und der Zielbereich nicht überlappen. Im Falle der Überlappung ist das Verhalten undefiniert.

Bibliothek zum Test von Zeichen: <ctype.h>

Das Argument der folgenden Funktionen ist jeweils ein `int` dessen Wert entweder `EOF` oder als `unsigned char` darstellbar sein muß. Der Rückgabewert ist ein `int`, wobei dieser gleich Null ist, wenn das Argument `c` die Bedingung nicht erfüllt, ansonsten ist er ungleich Null.

`islower(c)`	Kleinbuchstabe
`isupper(c)`	Großbuchstabe
`isalpha(c)`	Klein- oder Großbuchstabe
`isdigit(c)`	Dezimalzahl
`isalnum(c)`	Klein- oder Großbuchstabe oder Dezimalzahl
`iscntrl(c)`	Control-Zeichen
`isgraph(c)`	druckbares Zeichen außer space
`isprint(c)`	druckbares Zeichen mit space
`ispunct(c)`	druckbares Zeichen außer space, Buchstabe und Ziffern
`isspace(c)`	space, formfeed, newline, carriage return, tab, vertical tab
`isxdigit(c)`	Hexadezimalzahl

Achtung: Die deutschen Umlaute und das "ß" sind keine Buchstaben im obigen Sinne!!

In diesem Headerfile sind außerdem noch zwei Funktionen zur Konvertierung von Buchstaben enthalten:

`int tolower(int c)`	konvertiert `c` zu Kleinbuchstaben
`int toupper(int c)`	konvertiert `c` zu Großbuchstaben

Für die Bearbeitung von Zeichenketten gibt es eine umfangreiche Funktionsbibliothek:

Bibliothek mit String-Funktionen: <string.h>

Wie schon der Name sagt, werden in diesem Header Funktionen zur Manipulation von Strings deklariert. Dabei werden folgende Typkonventionen benutzt:

```
char *s, *t;
const char *cs, *ct;
int c;
size_t n;
```

`char *strcpy(s,ct)`	kopiert `ct` zu `s` inklusive dem abschließenden `'\0'`. Zurückgegeben wird `s`.

`char *strncpy(s,ct,n)`	kopiert höchstens `n` Zeichen des Strings `ct` zu `s` und liefert `s` zurück, wobei wie üblich mit `'\0'` terminiert wird.
`char *strcat(s,ct)`	hängt `ct` an `s` an und liefert `s` zurück.
`char *strncat(s,ct,n)`	hängt höchstens `n` Zeichen von `ct` an `s` an, terminiert mit `'\0'` und liefert `s` zurück.
`int strcmp(cs,ct)`	vergleicht `cs` und `ct`. Der Rückgabewert ist 0, wenn die beiden Strings identisch sind, negativ, wenn `cs<ct` ist und positiv, wenn `cs>ct` ist, wobei das < und > im lexikalischen Sinne verstanden wird, d.h. es wird Buchstabe für Buchstabe verglichen und geprüft, welcher als erster im Alphabet steht, wobei wiederum Großbuchstaben vor Kleinbuchstaben kommen (ASCII-Code).
`int strncmp(cs,ct,n)`	Im wesentlichen dasselbe wie zuvor, nur daß diesmal höchstens `n` Zeichen verglichen werden.
`char *strchr(cs,c)`	liefert Pointer zum ersten Auftreten von `c` in `cs` zurück, oder `NULL` wenn `c` nicht auftritt.
`char *strrchr(cs,c)`	liefert Pointer zum letzten Auftreten von `c` in `cs` zurück, oder `NULL` wenn `c` nicht auftritt.
`char *strstr(cs,ct)`	liefert Pointer zum ersten Auftreten des Strings `ct` in `cs` zurück oder `NULL` wenn `ct` nicht auftritt.
`size_t strlen(cs)`	liefert die Länge des Strings `cs` zurück.

Die Eingabefunktion gets

```
char *gets(char *s);
```

"gets" liest die nächsten Zeichen von stdin bis zum nächsten Newline-Character ('\n') bzw bis das Dateiende erreicht ist. Die gelesenen Zeichen werden in dem durch "s" referierten String ohne eventuell gelesenes '\n' abgelegt. An s wird automatisch Das Stringende-Zeichen '\0'angehängt. Funktionswert ist die Anfangsadresse des Strings "s" oder ein NULL-Pointer bei Erreichen des Dateiendes ohne vorheriges Lesen von Zeichen. Im Fehlerfall wird ebenfalls ein NULL-Pointer zurückgegeben. Es wird nicht überprüft, ob genügend Speicherplatz für den String zur Verfügung steht.

Die Ausgabefunktion puts

```
int puts(const char *s);
```

"puts" gibt den durch "s" referierten String (ohne '\0'-Character!) nach stdout aus, wobei ein abschließendes '\n' angefügt wird. Funktionswert ist ein nicht-negativer Wert bei Fehlerfreiheit und EOF (-1) im Fehlerfall.

```
/* String einlesen und wieder ausgeben */
#include <stdio.h>

char str[100];

void main(void)
  {
  if (gets(str) != NULL)
    puts(str);
  else
     printf("Leereingabe\n");
  }
```

Die Eingabefunktion getchar

```
int getchar(void);
```

"getchar" liest das nächste Zeichen von stdin und gibt das gelesene Zeichen (als int-Wert zurück. Bei Eingabe des Fileendezeichens für Textdateien oder im Fehlerfall wird EOF (-1) zurückgegeben.

```
/* Zeichen kopieren von stdin nach stdout */
#include <stdio.h>

int ch;

void main(void)
  {
  while ((ch = getchar()) != EOF)
    putchar(ch);
  }
```

Die Ausgabefunktion putchar

```
int putchar(int c);
```

"putchar" gibt das Zeichen "c" (nach Umwandlung in unsigned char) nach stdout aus. Funktionswert: das ausgegebene Zeichen (als int-Wert) oder EOF (-1) im Fehlerfall.