

C-Kurs

Strings

Unsere nächste „Datenstruktur“

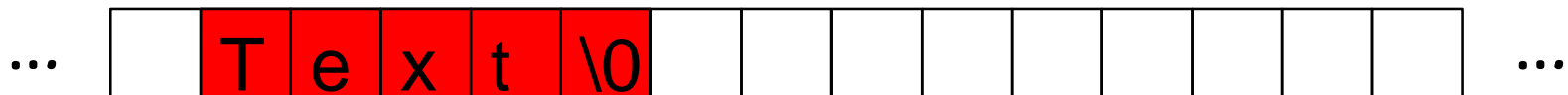
Zeichenketten / Strings

□ **Strings** sind char Arrays, die mit `'\0'` beendet werden

□ Beispiele:

- „Text“
- Formatstrings für printf
- Dateinamen
- Textdateien
- ...

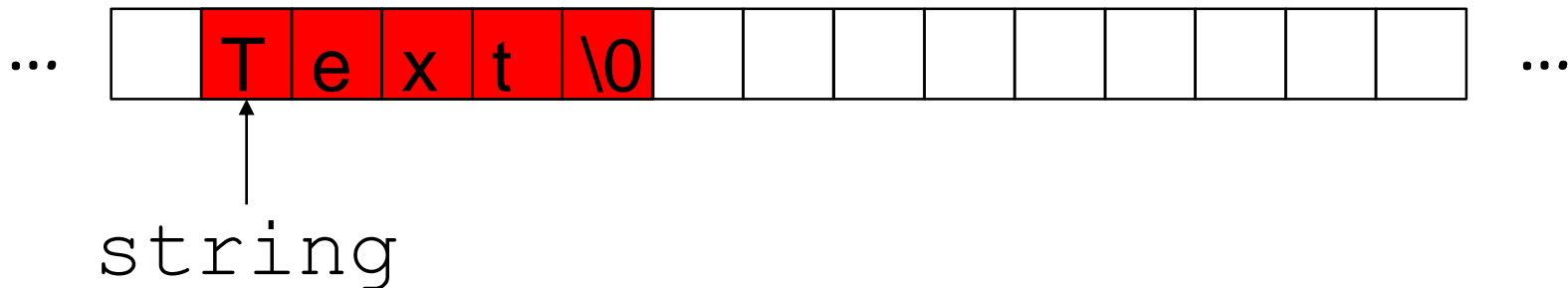
- ❑ **Strings** sind char Arrays, die mit `'\0'` beendet werden
- ❑ In C sind Strings eine Liste von Zeichen, d.h. eine Liste von char's, die hintereinander im Speicher stehen.
- ❑ Jedes Zeichen belegt ein Byte.
- ❑ Beispiel:



```
char * string = "Text";  
printf("Der Inhalt von string ist: %s\n",  
      string);
```

- ❑ **Strings** sind char Arrays, die mit `\0` beendet werden
- ❑ In C sind Strings eine Liste von Zeichen, d.h. eine Liste von char's, die hintereinander im Speicher stehen.
- ❑ „Stringvariablen“ sind Pointer, die auf den Start dieses Arrays zeigen:

```
char * string = "Text";
```



□ Hinweis:

Die Kodierung des Zeichensatzes kann unterschiedlich sein

- ASCII-Zeichensatz 7 Bit
- ASCII-Zeichensatz 8 Bit (PC, MS-DOS)
- ANSI-Zeichensatz (Windows)
- ...

Strings: Zusammenfassung

- ❑ Strings sind eine Folge von Einzelzeichen **char**
- ❑ String ist terminiert mit **`'\0'`**
- ❑ Speicherbedarf: Länge + 1 Byte
- ❑ String: Pointer auf Array von Elementen vom Type **char**
- ❑ Der leere String oder auch Nullstring: **NULL**
definiert in `stdio.h`

Einige Stringfunktionen

□ Ein paar der wichtigsten Stringgrundfunktionen

□ `#include <string.h>`

➤ `int strlen(char *s)`

Liefert Länge des String `s` ohne `'\0'`

➤ `char *strncpy(char *s1, char *s2, int n)`

Kopiert String `s2` nach String `s1`, **max `n` Zeichen**

▪ `char *strcpy(char *s1, char *s2)`

Kopiert String `s2` nach String `s1`, bis zum `s2` terminierenden NUL, liefert Ptr auf `s1`

▪ **Achtung: Pufferüberlauf Gefahr!**

Stringfunktionen

➤ `char *strncat(char *s1, char *s2, int n)`

Hängt Kopie von `s2` an `s1`, max. `n` Zeichen

- `char *strcat(char *s1, char *s2)`

Hängt Kopie von `s2` an `s1`, liefert Ptr auf `s1`

- **Achtung: Pufferüberlauf Gefahr!**

➤ `int strncmp(char *s1, char *s2)`

Vergleicht zeichenweise `s1` und `s2` liefert 0 bei Gleichheit
sonst Differenz `*s1 - *s2` bei erstem Unterschied,
max. `n` Zeichen

➤ `int strcmp(char *s1, char *s2)`

Vergleicht zeichenweise `s1` und `s2` liefert 0 bei Gleichheit
sonst Differenz `*s1 - *s2` bei erstem Unterschied

- ❑ C-Standardbibliothek bietet eine große Menge an Stringfunktionen an
- ❑ Darunter Konvertierungsfunktionen

`#include <stdlib.h>`

- `int atoi(char *s)`
wandelt String `s` in `int`
- `long atol(char *s)`
wandelt String `s` in `long`
- `double atof(char *s)`
wandelt String `s` in `double`

Stringfunktionen

- Weitere Konvertierungsfunktionen

- `#include <stdio.h>`

- `int sprintf(char *s, char *fmt, ...)`
formatierte Ausgabe auf String `s` wie `printf()`

Ausgabe mittels printf

Formatierte Ausgabe: printf

Aufruf: `printf(fmt, args)`

- ❑ `printf()` konvertiert und gibt die Parameter `args` unter Kontrolle des Formatstrings `fmt` auf `stdout` aus
- ❑ Der Formatstring `fmt` ist eine Zeichenkette
- ❑ Parameter `args` können auch fehlen
- ❑ Die Parameter `args` müssen den Typ haben, wie er im Formatstring `fmt` angegeben ist
- ❑ Beispiele
 - `printf("Hello world\n");`
 - `printf("Wert der Variablen i: %d\n", i);`
 - `printf("a(%d)+b(%d) ist: %d\n", a, b, a+b);`

□ Wichtige Formatzeichen:

<code>%c</code>	Einzelzeichen
<code>%d</code>	Integer
<code>%u</code>	Unsigned Integer
<code>%lu</code>	Unsigned Long
<code>%ld</code>	Integer
<code>%lld</code>	Integer
<code>%f</code>	Gleitkommazahl
<code>%lf</code>	Gleitkommazahl
<code>%s</code>	Zeichenkette/String

<code>char</code>
<code>int</code>
<code>unsigned int</code>
<code>long</code>
<code>long int</code>
<code>long long int</code>
<code>float</code>
<code>double</code>
<code>char *</code>

Formatierte Ausgabe: printf

Aufruf: `printf(fmt, args)`

□ Weitere Beispiele

➤ `printf("a(%d)/b(%d) ist: %d\n",
a, b, a/b);`

➤ `printf("a(%f)/b(%f) ist: %f\n",
a, b, a / (float) b);`

➤ `printf("Die Kodierung von %c ist %d\n",
'a', 'a');`

❑ Wichtige Sonderzeichen

`\n` **Newline, Zeilensprung**
`\t` **Tabulator**
`\0` **EOS - Endezeichen in String**

❑ Maskierung (Escaping) von reservierten Zeichen

`\'` einfaches Anführungszeichen `'`
`\"` doppeltes Anführungszeichen `"`
`%%` Prozentzeichen `%`
`\\` Backslash `\`

Wiederholung: Arrays

□ Ein **Array** (Feld):

- Ist eine Liste von Daten gleichen Typs
- Hat eine feste Länge
- Zugriff auf Arrayelemente mit Index in **[]**

□ Beispiele:

```
char sbuf[128];  
sbuf[0] = 'H';    // Arrayindexanfang 0!  
sbuf[1] = 'E';  
sbuf[2] = 'L';  
sbuf[3] = 'L';  
sbuf[4] = 'O';
```

Strings vs. Arrays

❑ Strings sind char Arrays

```
➤ char *s = "test";  
    char c = s[1]; // c = 'e';
```

❑ Aber es gibt wesentliche Unterschiede

- Strings müssen `\0` terminiert werden (d.h. 1 Zeichen länger)
- Arrays haben feste Länge im Gegensatz zu Strings

❑ Hinweis: Man speichert Strings in Arrays.
Muss aber mit der Länge vorsichtig sein.
Gefahr: Bufferoverflows
(Überschreiben von anderem Speicher)

Kommandozeilenparameter

Kommandozeilenparameter

Jedes C-Programm startet mit der Funktion `main`

```
int main (int argc, char *argv[]) {}
```

Main ist vom Typ `int` und hat bis zu drei Parameter:

- ❑ `int argc`

Anzahl von Kommandozeilenparametern

- ❑ `char *argv[]`

Array von Strings \Rightarrow Kommandozeilenparameter

- ❑ `char *envv[] // optional`

Array von Strings \Rightarrow Umgebungsparameter

- ❑ Rückgabewert wird vom Betriebssystem ausgewertet

 - Konvention Wert 0 bedeutet Programm zeigt keinen Fehler an

 - Konvention Werte $\neq 0$ bedeuten Programm hat Fehler erkannt

Kommandozeilenparameter 2

□ Beispiel:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
// entspricht      char **argv
    int i;
    for(i = 0; i < argc; i++) {
        printf("%d: %s\n", i, argv[i]);
    }
    return(0);
}
```

Eingabe

Formatierte Eingabe: `scanf`

Aufruf: `scanf(fmt, args)`

- ❑ `scanf()` liest von `stdin` (üblicherweise Tastatur) und versucht die Eingabe unter Kontrolle des Formatstrings `fmt` auf die Parameter `args` abzubilden
- ❑ Der Formatstring `fmt` ist eine Zeichenkette mit Leerzeichen
- ❑ Der Parameter `args` darf nicht fehlen
- ❑ Die Parameter `args` müssen den selben Typ haben, wie sie im Formatstring `fmt` angegeben sind
- ❑ Beispiele:
 - `int a, b; scanf("%d %d", &a, &b);`
 - `float x; scanf("%f", &x);`
 - `char a; scanf("%c", &a);`

Formatierte Eingabe: scanf

Aufruf: `int scanf(fmt, args)`

❑ RETURN VALUE von `scanf()`

These functions return the number of input items successfully matched and assigned, which can be fewer than provided for, or even zero in the event of an early matching failure.

❑ D.h. Der Rückgabewert von `scanf` gibt an

- Wenn erfolgreich: Wie viele Einträge erfolgreich gelesen wurden
- Wenn nicht erfolgreich: -1 (EOF == End Of File)

Formatierte Eingabe: `scanf`

Aufruf: `int scanf(fmt, args)`

❑ RETURN VALUE von `scanf()`

These functions return the number of input items successfully matched and assigned, which can be fewer than provided for, or even zero in the event of an early matching failure.

The value EOF is returned if the end of input is reached before either the first successful conversion or a matching failure occurs. EOF is also returned if a read error occurs, in which case the error indicator for the stream (see `ferror(3)`) is set, and `errno` is set indicate the error.