Konzepte des prozeduralen Programmierens

Stand September 2022

- Prof. Dr. Oliver S. Lazar / Christian Frank

8 Strings (Zeichenketten)

Strings in C

- für die Verarbeitung von einzelnen Zeichen verwenden wir den Datentyp **`char`**
- Strings sind aneinander gereihte Zeichen, folglich also ein **`char`**-Array

```
char string[] = "aber der Weise belächelt sie.";

printf("Der Kluge ärgert sich über die Dummheiten, %s\n", string);
```

```
Der Kluge ärgert sich über die Dummheiten, aber der Weise belächelt sie.
```

- Initialisierung eines **`char`**-Arrays erfolgt mit einer Zeichenkette in doppelten Hochkommas (keine geschweiften Klammern)
- Die Array-Größe ist gleich der Anzahl der Buchstaben des Initialisierungs-Strings + 1

Nullterminierte Strings

- ein String besteht in C aus einem **`char`**-Feld. Dieses Feld ist meist größer als der String selbst.

```
char string[100] = "Hallo";
```

- um das Ende einer Zeichenkette in einem char-Array zu bestimmen, werden Strings mit dem ASCII-Zeichen 0 abgeschlossen '\0',
  - Im Hauptspeicher steht nicht nur **`"Hallo"`**, sondern **`"Hallo\0"`**

- bei der Initialisierung ohne geschweifte Klammern wird das Ende-Zeichen automatisch gesetzt

- solche Zeichenketten mit abschließendem 0-Zeichen heißen ***nullterminierte Strings***

Nullterminierte Strings

- bei der Deklaration von String-Feldern sollte das Ende-Zeichen immer mit eingerechnet werden

```
// komplett gefuelltes Feld
char text[6] = "Hallo";
```

- Zeichenketten können auf diese Weise auch einfach abgeschnitten werden

```
char text[10] = "abcdefg";
printf("%s\n", text);
text[3] = '\0';
printf("%s\n", text);
```

```
abcdefg
abc
```

Zeichenketten anzeigen und einlesen

- **Anzeigen**
  - `char *satz="Programmieren macht Freude.";`
    - `// Zeichenkette mit * deklariert ist konstant`
  - `puts(satz);`
  - `printf("%s",satz);`
  - `printf("%20s",satz);` `// mindestens 20 Zeichen rechtsbündig`
  - `printf("%-20s",satz);` `// mindestens 20 Zeichen linksbündig`
  - `printf("%.5s",satz);` `// höchstens 5 Zeichen`

- **Einlesen**
  - `char zeile[81];`
  - `gets(zeile);` // lesen bis Newline (besser mit fgets)
    - `fgets(buffer, 80 , stdin);`
  - `scanf("%s",zeile);` // lesen bis Whitespace

String-Funktionen


```
#include<string.h>
```

String-Funktionen

String Länge mit `strlen`

```
size_t strlen(char *str);
```

- mit **`strlen`** kann man die Länge einer Zeichenkette bestimmen

```
int size = 0;
char string[] = {'F', 'O', 'M', '\0'};

size = strlen(string);

printf("\nLaenge von s: %d\n", size);
```

```
Laenge von s: 3
```

String Copy - Strings kopieren in C mit `strcpy` und `strncpy`

```
char *strcpy(char *dest, char *src);
```

- mit **`String Copy`** können wir den Inhalt eines Strings kopieren.
  - **`*dest`** ist Zeiger auf Ziel-Array
  - **`*src`** ist Zeiger auf Quell-Array
  - Rückgabewert ist **`char`** Zeiger auf Ziel-Array

```
char textA[5] = "abc";
char textB[5] = "abc";

printf("Text A: %s\nText B: %s\n\n", textA, textB);

// konstante Zeichenkette in String B kopieren
strcpy(textB, "xyz");
printf("Text A: %s\nText B: %s\n\n", textA, textB);

// char-Array B zu char-Array A kopieren
strcpy(textA, textB);
printf("Text A: %s\nText B: %s\n\n", textA, textB);
```

```
Text A: abc
Text B: abc

Text A: abc
Text B: xyz

Text A: xyz
Text B: xyz
```

String Copy - Strings kopieren in C mit `strcpy` und `strncpy`

```
char *strncpy(char *dest, char *src, int n);
```

- mit **`strncpy`** kopiert man **n** Zeichen von **src** nach **dest**.
  - ist die Länge des zu kopierenden Strings kleiner als die Länge des Quell-Strings, wird das Ende-Zeichen im Ziel-String nicht gesetzt. Dies muss also manuell gesetzt werden.

```
char textA[10] = "123456";
char textB[10];

// kopiere in textB 3 Zeichen von textA
strncpy(textB, textA, 3);

// Ende-Zeichen setzen
textB[3] = '\0';

printf("Text A: %s\nText B: %s\n\n", textA, textB);
```

```
Text A: 123456
Text B: 123
```

Strings verketten in C mit strcat und strncat

```
char *strcat(char *dest, char *src);
```

- mit **strcat** kann man zwei Zeichenketten verketten.

```
char textA[10] = "abc";
char textB[5] = "xyz";


printf("Text A: %s\nText B: %s\n\n", textA, textB);


// haenge Zeichenkette textB an textA an
strcat(textA, textB);

printf("Text A: %s\nText B: %s\n\n", textA, textB);
```

```
Text A: abc
Text B: xyz

Text A: abcxyz
Text B: xyz
```

String-Funktionen

Strings verketten in C mit strcat und strncat

```
char *strncat(char *dest, char *src, int n);
```

- mit **`strncat`** werden `n` Zeichen aus `src` an `dest` angehängt.

```
char textA[10] = "abc";
char textB[10] = "defxyz";


printf("Text A: %s\nText B: %s\n\n", textA, textB);


// haenge 3 Zeichen aus textB der Zeichenkette textA an
strncat(textA, textB, 3);

printf("Text A: %s\nText B: %s\n\n", textA, textB);
```

```
Text A: abc
Text B: defxyz

Text A: abcdef
Text B: defxyz
```

Aufg. 08.02

- Der Benutzer soll eine Ziffer für einen Tag eingeben und den Tagnamen ausgegeben bekommen; z. B. 1 → Montag etc.; speichere die Tagesbezeichnungen in einem array.

  - Codeelemente
    - `char* wochentage[7];`
    - `wochentage[0] = "Montag";`

Aufg. 08.03

- Erstelle und initialisiere ein Array mit 5 Vornamen und ein Array mit 5 Nachnamen. Gib dann die zueinandergehörenden Vor- und Nachnamen so aus, dass die Vornamen linksbündig und die Nachnamen rechtsbündig stehen. Fülle die Lücke so mit Unterstrichen aus, dass jede Zeile genau 80 Zeichen hat.

  - Codeelemente
    - strlen
    - strcat
    - strcpy
    - char ausgabe[81]

Strings vergleichen mit strcmp

```
int strcmp(char *str1, char *str2);
```

- mit **`strcmp`** können zwei Zeichenketten verglichen werden.
- folgende Rückgabewerte sind möglich:
  - 0 die Strings sind gleich
  - 1 das erste ungleiche Zeichen in str1 ist größer als in str2
  - -1 das erste ungleiche Zeichen in str1 ist kleiner als in str2
- Beispiel auf der folgenden Folie

Strings vergleichen mit strcmp

```
char str1[] = "aaXaa";
char str2[] = "aaYaa";

// vergleiche str1 mit str2, X < Y, also -1
printf("Vergleich str1 mit str2: %d\n", strcmp(str1, str2));

// vergleiche str2 mit str1, Y > X, also 1
printf("Vergleich str2 mit str1: %d\n", strcmp(str2, str1));

// setze Strings gleich
strcpy(str2, str1);

if(strcmp(str1, str2) == 0) {
    printf("str1 ist gleich str2!\n");
}
```

```
Vergleich str1 mit str2: -1
Vergleich str2 mit str1: 1
str1 ist gleich str2!
```

String-Suche mit `strstr`

```
char *strstr(char *haystack, char *needle);
```

- „Die Nadel im Heuhaufen suchen“
  - mit strstr kann man das Vorkommen einer Zeichenkette (needle) in einer anderen (haystack) überprüfen

```
char haystack[] = "aa123aa";
char needle[] = "123";

if(strstr(haystack, needle)) {
    printf("Haystack enthaelt '123'\n");
}
```

```
Haystack enthaelt '123'
```

Zeichen-Suche in einem String mit `strchr`

```
char *strchr(char *s, int c);
```

- mit **`strchr`** kann man ein Zeichen in einem String suchen
- das zu suchende Zeichen wird mit dem Parameter **`c`** als ASCII-Code übergeben
- die Rückgabewerte:
  - **NULL**, wenn das Zeichen nicht gefunden wurde
  - Adresse des **ersten** gefundenen Zeichens, wenn es vorkommt

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

if(strchr(string, 'X')) {
    printf("String enthaelt ein X\n");
}
```

```
String enthaelt ein X
```

Zeichen-Rückwärts-Suche in einem String mit `strrchr`

```
char *strrchr(char *s, int c);
```

- mit **`strrchr`** kann man ein Zeichen in einem String suchen
- das zu suchende Zeichen wird mit dem Parameter **`c`** als ASCII-Code übergeben
- die Rückgabewerte:
  - **NULL**, wenn das Zeichen nicht gefunden wurde
  - Adresse des **ersten** gefundenen Zeichens vom Ende der Zeichenkette aus, wenn es vorkommt

Zeichen-Rückwärts-Suche in einem String mit `strrchr`

- Beispiel

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

printf("vorher: %s\n", string);

char *c1 = strchr(string, 'X');
char *c2 = strrchr(string, 'X');

*c1 = 'A';
*c2 = 'B';

printf("nachher: %s\n", string);
```

```
vorher: aaXaaXaa

nachher: aaAaaBaa
```

Aufgabe

Aufg. 08.04

- Benutze die String-Funktionen der Bibliothek <string.h>, um aus einem vollständigen Pfad in Form eines Strings das Verzeichnis, den Dateinamen und die Extension der Datei zu ermitteln. Lautet z.B. der gesamte Pfad:

```
C:\Eigene Dateien\FOM\C-Code\main.c
```

dann soll das Programm folgendes extrahieren:

```
Extension:          c
Dateiname:          main.c
Verzeichnis:C:\Eigene Dateien\FOM\C-Code
```

- Vorgeschlagene Codeelemente
  - char string[] = "C:\\Eigene Dateien\\FOM\\C-Code\\main.c";
  - `strrchr()`
  - Setze jeweils den Pointer auf die richtige Stelle
  - Verzeichnis: Kürze den Dateipfad mit '\0' an der richtigen Stelle

Klausur WS14/ Aufg. 10
Klausur SS-15-Nachschreib/ Aufg. 10
Klausur WS-18-19 Aufg. 10

String-Funktionen

String zerteilen und splitten in C mit `strtok`

```
char *strtok(char *string, char *delimiters);
```

-

```
char string[] = "FOM,45141";
char delimiter[] = ",;";
char *ptr;

// initialisieren und ersten Abschnitt erstellen
ptr = strtok(string, delimiter);

printf("Abschnitt gefunden: %s\n", ptr);

// naechsten Abschnitt erstellen
ptr = strtok(NULL, delimiter);
printf("Abschnitt gefunden: %s\n", ptr);
```

```
Abschnitt gefunden: FOM
Abschnitt gefunden: 45141
```

String zerteilen und splitten in C mit `strtok` und `while`-Schleife

```
char string[] = "FOM,Neutron;45219;DE";
char delimiter[] = ",;";
char *ptr;

// initialisieren und ersten Abschnitt erstellen
ptr = strtok(string, delimiter);

while(ptr != NULL) {
    printf("Abschnitt gefunden: %s\n", ptr);
    // naechsten Abschnitt erstellen
    ptr = strtok(NULL, delimiter);
}
```

```
Abschnitt gefunden: FOM
Abschnitt gefunden: Neutron
Abschnitt gefunden: 45219
Abschnitt gefunden: DE
```

Typumwandlungen

- Kommen Zahlen in Strings vor, so kann man Umwandlungsfunktionen aus der Bibliothek **<stdlib.h>** nutzen
  - String zu int: **atoi**
  - String zu long int: **atol**
  - String zu double: **atof**

```
char charNumber1[] = "100";
char charNumber2[] = "12.5";

int number1 = atoi(charNumber1);
double number2 = atof(charNumber2);

double sum = number1 + number2;

printf("%d + %.2f = %.2f\n", number1, number2, sum);
```

```
100 + 12.5 = 112.5
```

Aufg. 08.05

- Schreibe eine Funktion, welcher ein Zeiger auf einen String (char Array) übergeben werden kann. Die Funktion soll alle Kleinbuchstaben in Großbuchstaben umwandeln. Implementiert werden soll die Funktion in einer **for** Schleife mit höchstens einem **if**. Verwende diesen Funktions-Prototyp:

```
void stringToUpper(char *string);
```

- Vorgeschlagene Codeelemente:
  - `char string[] = "Schweigen ist ein Zaun um Weisheit";`
  - `if(string[i] >= 'a' && string[i] <= 'z')`
    - Großbuchstabe= ASCII-Code -32 oder
    - die Funktion `toupper(int ch)` aus `<ctype.h>`

Aufg. 08.06

- Es soll eine Funktion geschrieben werden, welche in einem String einen String ersetzt. Der gesuchte String und der Ersetzungs-String können hierbei eine unterschiedliche Länge haben. Dafür wäre die Verwendung der dynamischen Speicherverwaltung hilfreich.

```
char * stringReplace(char *search, char *replace, char *string);
```

```
Vorher:  In der Ruhe liegt die Kraft.
Nachher: In der Stille liegt die Kraft.
```

- Vorgeschlagene Codeelemente:
  - mit temporärer Kopie arbeiten
    - `tempString = (char*) malloc(strlen(string) * sizeof(char));`
  - **`strstr()`**

Aufg. 08.07

- Gegeben sei folgende Komma-separierte Zeichenkette:
  - `char string[] = "Paris 111,Los Angeles 6,London 16,Rom 28";`
- Teile diese Zeichenkette beim Trennzeichen Komma in einzelne Abschnitte.
- Extrahiere aus jedem Abschnitt die Zahlenwerte.
- Wandel die extrahierten Zahlenwerte in Integer-Zahlen um und berechne die Summe.
- Vorgeschlagene Codeelemente:
  - `ptr = strtok(string, delimiter);`
  - `strrchr(ptr,' ')` **`und`** `atoi(ptr)`

Klausur WS17-18-Nachschreib/ Aufg. 6