Strings

Für Strings (Zeichenketten, "Texte") gilt in C:

- Strings sind *Arrays von einzelnen Zeichen*, also Arrays vom Typ **`char`**.
  Beispiel: **`char input[82];`**
  Man kann daher mit **`[ ]`** einzelne Zeichen eines Strings lesen oder schreiben:
  ```
  if (argv[1][0] == '-') /* Beginnt argv[1] mit einem Minus? */
  input[i] = toupper(input[i]);
        /* schreibe das i-te Zeichen von "input" groß */
  ```
- String-Konstanten stehen in *doppelten* Anführungszeichen: **`"Franz"`**
  Einzelne Zeichen (wenn sie kein String, sondern ein **`char`** sein sollen) stehen in *einfachen* Anführungszeichen: **`'X'`**
- Der Wert eines einzelnen **`char`** ist eigentlich eine Zahl, nämlich der ASCII-Wert des Zeichens. Beispiel: **`'0'`** bis **`'9'`** ist 48 bis 57, **`'A'`** bis **`'Z'`** ist 65 bis 90.
  Man kann damit sogar rechnen: Wenn **`c`** ein **`char`** ist und eine Ziffer enthält, so liefert **`c - '0'`** den numerischen Wert der Ziffer (0 bis 9). Warum?
- C speichert nirgends, wie viele Zeichen ein String wirklich enthält, aber es markiert das *Ende jedes Strings* mit **`'\0'`** (dem **`char`** mit dem Wert 0, nicht **`'0'`**). Das hat viele Folgen:
  - Wenn man einen String Zeichen für Zeichen durcharbeitet, geht man nicht bis zur Länge des Arrays (d.h. bis zum letzten Element des Arrays), sondern bis zu dem Element, das **`'\0'`** enthält.
    Der "gültige" Teil eines Strings bis zum **`'\0'`** ist meistens kürzer als das Array. Die Elemente nach dem **`'\0'`** bis zum Ende des Arrays sind undefiniert bzw. werden ignoriert.
  - Wenn man einen String deklariert, muss man das Array ***mindestens ein Element größer machen***, als man darin ***maximal*** Zeichen speichern will, weil das **`'\0'`** hinten dran muss immer auch noch Platz haben (zu groß schadet nichts, außer dass es Platz kostet).
    Bei Eingabezeilen usw. müssen es sogar *zwei* Elemente mehr als die maximale Anzahl von Zeichen sein: Auch das **`'\n'`** braucht einen Platz.
  - Wenn man selber einen String Zeichen für Zeichen zusammenbastelt, darf man nicht vergessen, *hinter dem letzten Zeichen ein* **`'\0'`** *anzuhängen*. Sonst weiß C nicht, wo der String aus ist, und stürzt womöglich ab.
  - Mit **`str[0] = '\0';`** initialisiert man einen String **`str`** auf den Leerstring **`""`**, und mit **`if (str[0] == '\0')`** kann man prüfen, ob **`str`** leer ist.
- Die vordefinierte Funktion **`strlen`** (aus **`string.h`**) liefert die Länge eines Strings (die benutzen Zeichen, nicht die Größe des Arrays), z.B. **`len = strlen(input);`**
  Die "sichtbaren" Zeichen stehen in **`input[0]`** bis **`input[len - 1]`**, **`input[len]`** ist **`'\0'`**.
- Die Funktion **`strcmp`** vergleicht zwei Strings alphabetisch (Details siehe Übung).
- Die Funktion **`strcpy`** kopiert einen String: **`strcpy(nach, von);`**
  `strcat` hängt einen String hinten an einen anderen an: **`strcat(nach, von);`**
  Achtung: *Das Array* **`nach`** *muss lang genug sein, sonst stürzt das Programm ab!*
- In **`ctype.h`** gibt es ein paar nützliche Funktionen für einzelne Zeichen:
  `toupper` und **`tolower`** zur Umwandlung zwischen Groß- und Kleinbuchstaben und einige Funktionen, die einen **`char`** prüfen und "wahr" oder "falsch" liefern:
  `isdigit`, **`isalpha`**, **`isupper`**, **`islower`**, **`isspace`**, …