4.4 Datenstrukturen/Strings

4.4.1 Aufgabe

Für einen Rechtschreibtrainer soll eine gegebene Zeichenkette so umgewandelt werden, dass alle Kommazeichen entfernt werden. Aus der Zeile „Das, was ich meine, sage ich auch.“ soll die Zeile werden: „Das was ich meine sage ich auch.“

4.4.2 Beispiel

Auf einer Website für Programmieranfänger findet sich folgendes Beispiel (`src/leernachplus.c`):

```
1  #include <stdio.h>
2  #include <string.h> /* fuer strlen()-Funktion */
3  int main(void)
4  {
5      char ausgabe[81];
6      char eingabe[] = "Das␣ist␣das␣Haus␣vom␣Nikolaus.";
7      int eingabe_laenge;
8      int lauf;
9      eingabe_laenge = strlen(eingabe);
10
11     for(lauf=0; lauf<eingabe_laenge; ++lauf)
12     {
13         if(eingabe[lauf]=='␣')
14         {
15             ausgabe[lauf]='+';
16         }
17         else
18         {
19             ausgabe[lauf]=eingabe[lauf];
20         }
21     }
22     ausgabe[eingabe_laenge]='\0';
23     printf("%s\n", ausgabe);
24     return 0;
25 }/* main */
```

In Zeile 5 findet man die Vereinbarung einer Zeichenkette. Es ist die Kombination aus

- Datentyp `char`
- Datenstruktur Array

In der Variable `ausgabe` kann man also maximal 81 Zeichen unterbringen.

4.4.3 Initialisierung

In Zeile 6 wird eine weitere Variable vereinbart. Hier kann die Angabe der Feldgröße weggelassen werden, weil sie durch die anschließende Initialisierung festgelegt wird.

4.4.4 Zeichenkettenkonstante

In derselben Zeile sieht man eine Besonderheit für C: Man kann eine Zeichenkette direkt mit Anführungszeichen initialisieren. Der Inhalt der konstanten Zeichenkette `"Das ist ..."` wird Zeichen für Zeichen in das Feld `eingabe` kopiert. Die konstante Zeichenkette `"Das ist ..."`

Index	0	1	2	3
Inhalt	`'D'`	`'a'`	`'s'`	`'\0'`

Tabelle 1: Initialisierte Zeichenkette

ist altbekannt, z.B. von `printf("Das ist ...")`. Diese Form von Konstante (Zeichenkettenkonstante) gibt es nur für den Typ `char[]` (und für `w_char[]` für UNICODE), nicht aber für `int[]` oder `double[]`.

4.4.5 Terminator

Eigentlich hätte man eine andere Form der Initialisierung erwartet können:

```
1   char eingabe[]={'D','a','s'};
```

Damit diese beiden Initialisierungen im Ergebnis gleichwertig sind, muss noch etwas hinzugefügt werden:

```
1   char eingabe[]={'D','a','s','\0'};
```

Das angefügte Zeichen ist das Zeichen mit dem ASCII-Code null. Man hätte dafür auch einfach `0` schreiben können. Jetzt sind beide Initialisierungen im Ergebnis gleich (Tabelle 1). Aber wozu dient diese Null am Ende der Zeichenkette? Sie dient zum Bestimmen des Endes der Zeichenkette. Man nennt sie daher auch *Terminator*.

Zum Beispiel soll in einem Programm eine Variable für Vornamen 30 Zeichen enthalten. Nun wird der Vorname `Hans` eingegeben, der aber nur vier Zeichen braucht. Bei der späteren Ausgabe des Vornamens sollen nur diese vier Buchstaben ausgegeben werden. Dann wird hinter das `s` von `Hans` der Terminator gesetzz. Die Position der Null zeigt fremden Funktionen (wie `printf()`), wie viele Zeichen meine Zeichenkette momentan enthält, nämlich genau vier.

Wenn in irgendeiner Anwendung einmal die Null am Ende der Zeichenkette *nicht* eingefügt werden soll, muss man die Länge der Zeichenkette ohne die Null explizit angeben:

```
1   char eingabe[3]="Das";
```

Eine der fremden Funktionen, von denen die Rede war, ist `strlen()` in Zeile 9. Diese Funktion sieht nach, an welcher Stelle das Nullzeichen steht und ermittelt daraus die Länge.

4.4.6 Verarbeitung

In der Schleife erfolgt Zeichen für Zeichen der Aufbau der zweiten Zeichenkette (Zeilen 11–21). Hierzu wird, wie bei anderen Arrays auch, eine Zählschleife benutzt.

4.4.7 Ausgabe

In Zeile 22 wird der Terminator an die Ausgabe-Zeichenkette angehängt, damit die anschließende `printf()`-Funktion weiß, wie weit sie ausgaben soll. In Zeile 23 folgt die Ausgabe. Für die Ausgabe von Zeichenketten kann man zwischen drei Formen wählen:

```
1   puts(ausgabe);
2   printf(ausgabe);
3   printf("%s", ausgabe);
```

Die erste Form mit `puts()` gibt die Zeichenkette aus und hängt ein `'\n'`-Zeichen an. Die zweite Form ist noch vom Hello-World-Programm bekannt und für konstante Zeichenketten in Ordnung. Problematisch ist sie nur, wenn die Zeichenkette ein Prozent-Zeichen enthält; dann erwartet die `printf()`-Funktion weitere Parameter, die es aber nicht gibt und bewirkt eventuell einen Programmabsturz. Daher sollte man diese Form bei Variablen vermeiden. Deshalb sollte man die dritte Form (oder die erste) verwenden.

4.4.8 Lösung

```
1  #include <stdio.h>
2  #include <string.h> /* fuer strlen()-Funktion */
3  int main(void)
4  {
5      char ausgabe[81];
6      char eingabe[] = "Das␣was␣ich␣meine,␣sage␣ich.";
7      int eingabe_laenge, lauf;
8
9      eingabe_laenge = strlen(eingabe);
10     for(lauf=0; lauf<eingabe_laenge; ++lauf)
11     {
12         if(eingabe[lauf]==',')
13         {
14             ausgabe[lauf]='␣';
15         }
16         else
17         {
18             ausgabe[lauf]=eingabe[lauf];
19         }
20     }
21     ausgabe[eingabe_laenge]='\0';
22     printf("%s\n", ausgabe);
23     return 0;
24 }/* main */
```

4.4.9 Eingabe von Zeichenketten

Die Eingabe von Zeichenketten ist in C leider nicht ganz einfach, da mit den gleichen Funktionen die Eingabe von der Tastatur und die Eingabe aus einer Datei realisiert wurde. Zudem hat man es in C mit *gepufferter Eingabe* zu tun, d.h., nur Eingaben, die mit einem Zeilenende (`'\n'`) abgeschlossen wurden, können von den Funktionen gelesen werden.

4.4.9.1 `gets` nicht benutzen Die Funktion `gets()`, die ursprünglich für die Eingabe von Zeichenketten vorgesehen war, ist defekt:

```
1      char zk[21];
2      gets(zk);
```

Sobald man hier mehr als 20 Zeichen eingibt, werden andere Variablen im Speicher davon überschrieben und das Programm kann abstürzen oder unerwartete Aktionen bewirken. Deshalb darf man `gets()` niemals verwenden!

4.4.9.2 Mit `scanf` ein Wort einlesen Mit der Funktion `scanf()` und dem Platzhalter `%s`ist eine Eingabe von Zeichenketten möglich.

```
1      char zk[21]="";
2      scanf("%20s", zk);
```

- Zeile 1: Die Initialisierung ist nötig. Falls der Benutzer nichts eingibt, behält `scanf()` den alten Inhalt von `zk`. Und dann ist es sinnvoll, wenn man schon vorher einen vernünftigen Standardwert (*Default*-Wert) in `zk` hat.

- Zeile 2, Formatstring: Mit der Zahl 20 zwischen Prozentzeichen und `s` teilt man scanf() mit, dass man maximal 20 Zeichen einlesen möchte. Mit Terminator braucht man dann 21 Zeichen für die Zeichenkette.
- Zeile 2, Parameter `zk`: Vor dem Namen der Variablen `zk` steht hier *kein* &-Zeichen (Kaufmanns-Und). Das liegt daran, dass `zk` ein Array ist. Und für Arrays gelten in C besondere Gesetze (dazu später mehr).

Die Sache hat noch einen Fehler: `scanf()` mit Platzhalter liest immer nur ein Wort aus dem Tastaturpuffer ein. Falls der Benutzer mehr als ein Wort eingegeben hat, verbleibt der Rest im Tastaturpuffer. Beim nächsten Aufruf von `scanf` wird das nächste Wort eingelesen. Bei jedem Leerzeichen, Tabulator oder `'\n'` beendet `scanf()` mit Platzhalter seine Arbeit.

4.4.9.3 Mit `scanf` eine Zeile einlesen `scanf()` mit Platzhalter liest also immer nur ein Wort. Das ist aber meistens nicht das, was man will. Deshalb gibt es einen weiteren Platzhalter, der es erlaubt, Zeichen einzulesen, die aus einer Gruppe von Zeichen stammen. Statt `%s` verwendet man die eckigen Klammern `%[]`:

```
1  char zk[21]="";
2  scanf("%20[ABC]", zk);
3  getchar();
```

- Zeile 2, Formatstring: Anstelle des `s` in `%s` steht hier ein Paar eckiger Klammern mit einer Menge von Zeichen darin. Das `s` fehlt hier deshalb!
- Zeile 2, Parameter `zk`: Auch hier ist wieder kein &-Zeichen (Kaufmanns-Und).
- Hier werden alle Zeichen eingelesen (maximal 20), die A, B oder C heißen. Nach Eingabe eines anderen Zeichens kehrt die Funktion zurück.
- Zeile 3: Dieses andere Zeichen liegt nun noch im Eingabepuffer und wird mit `getchar()` gelesen, damit es beim nächsten Aufruf nicht schon wieder gelesen wird und die Eingabe beendet.

Dieses Beispiel lohnt sich also für alle Zeilen, die nur aus A, B und C bestehen.

Wenn man eine ganze Zeile auch mit anderen Zeichen einlesen will, muss man in die eckigen Klammern alle Buchstaben (groß und klein), alle Ziffern und alle Satzzeichen einbauen. Nur das Zeilenendezeichen lässt man weg, denn das soll das Ende der Zeile markieren:

```
1  char zk[21]="";
2  scanf("%20[]A-Za-z0-9;,:._!\"$%&/(){}\\?+#<>|-]", zk);
3  getchar();
```

- Zeile 2, Formatstring: `A-Z` meint alle Zeichen von A bis Z, also alle Großbuchstaben. Ebenso meint `a-z` alle Kleinbuchstaben und `0-9` alle Ziffernzeichen. Das Zeichen `]` muss ganz am Anfang stehen, damit es nicht als Ende der eckigen Klammern interpretiert wird, das Zeichen `-` steht am Schluss, damit es nicht als Bereichs-Zeichen (wie in `A-Z`) verstanden wird, und das Anführungszeichen muss wie der Backslash `\` durch einen Backslash maskiert werden.

Das war kompliziert! Und es kann immer noch sein, dass man ein Zeichen vergessen hat... Deshalb gibt es eine viel einfachere Lösung:

```
1  char zk[21]="";
2  scanf("%20[^\n]", zk);
3  getchar();
```

- Zeile 2, Formatstring: Das `'^'`-Zeichen bewirkt eine Verneinung. Nun werden alle die Zeichen eingelesen, die *nicht* in der eckigen Klammer angegeben sind, also alle außer dem `'\n'` Zeichen.
- Zeile 3: Das Zeichen `\n` liegt noch im Eingabepuffer und wird mit `getchar()` gelesen. Will man bei überlangen Zeilen die überschüssigen Zeichen einfach wegwerfen, dann ersetzt man diese Zeile durch **while**(getchar()!='\n'){}.

4.4.9.4 `scanf` und die maximale Eingabelänge Es ist sinnvoll, die Länge der Zeichenkette nicht durch eine *wörtliche Konstante* (z.B. `20`) festzulegen, die man an verschiedenen Stellen im Programm wiederholen muss, sondern durch eine am Beginn festgelegte *symbolische Konstante* (`SLAENGE`):

```
1 #define SLAENGE       20
2 #define SLAENGESTR "20"
3     char zk[SLAENGE+1]="";
4     scanf("%" SLAENGESTR "[^\n]", zk);
5     getchar();
```

- Zeile 1: Hier wird `SLAENGE` als 20 definiert.
- Zeile 3: Hier bekommt `zk` Speicherplatz für 21 Zeichen.
- Zeile 2: Leider muss man für `scanf()` außer `SLAENGE` noch eine zweite Konstante `SLAENGESTR` anlegen, die denselben Wert als Zeichenkette `"20"` enthält.
- Zeile 4: Der Präprozesor macht vor dem Compilieren aus den drei Zeichenketten eine:

```
1 "%"  "20"  "[^\n]"
```

  Wird zu:

```
1 "%20[^\n]"
```

  Das wirkt dann als Formatstring. Es werden maximal 20 Zeichen gelesen.

4.4.9.5 `fgets` Eine häufig benutzte Funktion zum Einlesen von Zeilen ist `fgets()`. Mit `fgets()` kann man auch aus Dateien einlesen. Leider liest `fgets()` nicht nur die gewünschte Zeile ein, sondern auch den Zeilenumbruch `'\n'` und speichert ihn (wie auch den Terminator) ab. Die Zeichenkette braucht in diesem Fall also für zwei zusätzliche Zeichen Platz. Außerdem muss man den Zeilenumbruch wieder löschen, wenn man die Zeichenkette benutzen will. Hier ein kurzes Rezept für einfache Fälle:

```
1 #include <string.h>
2 #define SLAENGE 20
3     char zk[SLAENGE+2]="";
4     fgets(zk, SLAENGE+2, stdin);
5     zk[strcspn(zk, "\n")]='\0';
```

Mit `strcspn()` (deklariert in `<string.h>`) wird der Zeilenumbruch entfernt. Eventuell vorhandene überschüssige Zeichen können bei diesem einfachen Rezept allerdings nicht entfernt werden und werden beim nächsten Mal ganz normal eingelesen.