7
Zeichenketten (Strings) in C

Jörn Loviscach

Versionsstand: 25. September 2013, 18:10

Die nummerierten Felder sind absichtlich leer, zum Ausfüllen beim Ansehen der Videos: http://www.j3L7h.de/videos.html

Bitte hier notieren, was beim Bearbeiten unklar geblieben ist

1 Grundlagen

Zeichenketten [Strings] werden in C als Arrays von `char` gespeichert. Statt dass man die Länge der Zeichenkette speichert, setzt man hinter das letzte Zeichen den Wert 0: „null-terminated string“. Mann könnte eine solche Zeichenkette im Prinzip so bauen:

1

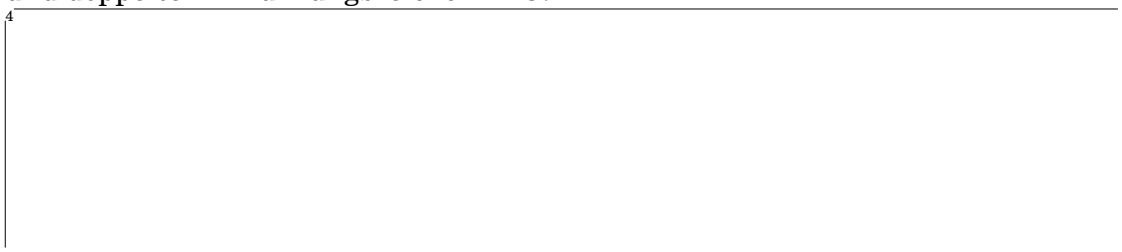
Oder anschaulicher:

[2]


Oder in der dann üblichen Form:

[3]


Die doppelten Anführungszeichen sagen dem Compiler, dass er eine Zeichenkette erzeugen soll – und zwar mit der selbstverständlichen Null am Ende, auch wenn man die in diesem Programmcode nicht sieht. Der Unterschied zwischen einfachen und doppelten Anführungszeichen in C:

[4]


Achtung: In anderen Programmiersprachen können die Bedeutungen der Anführungszeichen ganz andere sein.

Suchen aller Buchstaben `a` in einer Zeichenkette und Ersetzen durch `e`:

[5]


Demo auch mit der Speicheransicht in der Entwicklungsumgebung.

Nicht die Länge des Strings zu speichern, sondern eine Null ans Ende zu setzen, sah am Anfang wie eine gute Idee aus. Heute ist man klüger. In Java und C# macht man es anders; in C++ kann und sollte man es anders machen; und auch in C gibt es inzwischen einige Funktionen, die vorsichtiger mit Zeichenketten umgehen.

Das wesentliche Problem mit der Null am Ende ist, dass sie fehlen kann und dass die Zeichenkette damit in den nachfolgenden Speicher hineinzuragen scheint – eine Möglichkeit für schwer zu findende Fehler und ein wesentliches Einfallstor für Buffer-Overflow-Angriffe.

Wenn das lateinische Alphabet mit den üblichen Akzenten und Umlauten nicht reicht, kann man Zeichenketten statt mit `char` alternativ mit dem in `<wchar.h>` deklarierten Typ `wchar_t` bilden. Der hat typischerweise 16 Bit, unter Linux und Mac OS X aber 32 Bit. Achtung: `wchar_t` benötigt andere Funktionen zum Kopieren usw. als `char`.

2 Standardfunktionen für Strings

In `<string.h>` sind einige übliche Funktionen für Zeichenketten deklariert. Demos einiger Beispiele:

2.1 Länge

`int n = strlen(a);` liefert die Länge der Zeichenkette `a`, also die Zahl der Zeichen vor dem ersten Auftauchen von 0. Demo: Wenn die 0 fehlt, passiert Unsinn.

Von dieser Funktion gibt es eine sicherere Variante `strnlen`, der man eine Maximallänge mitgeben kann.

2.2 Vergleich

`int v = strcmp(a, b);` liefert eine Zahl, die sagt, was die alphabetische Sortierung der beiden Zeichenketten `a` und `b` ist: Ist das Ergebnis negativ, steht die erste Zeichenkette im Alphabet vor der zweiten; ist das Ergebnis null, sind beide Zeichenketten gleich; ist das Ergebnis positiv, steht die erste Zeichenkette im Alphabet hinter der zweiten. Eigentlich geht es gar nicht um das Alphabet, sondern um die Sortierung in der Liste der Zeichencodes: A steht vor Z und das steht vor a.

Diese Funktion ist auch ein Weg, festzustellen, ob zwei Zeichenketten gleich sind. Der Vergleich `a == b` funktioniert für Zeichenketten in C nicht (in moderneren Sprachen dagegen schon). Was `a == b` in C tatsächlich bedeutet, sehen wir demnächst bei den Zeigern.

Von dieser Funktion gibt es eine sicherere Variante `strncmp`, der man eine Maximallänge mitgeben kann.

2.3 Kopie

`strcpy(a, b);` kopiert die Zeichenkette `b` in die Zeichenkette `a`. Es wird nicht geprüft, ob `a` dafür groß genug ist, was gefährlich ist. Die etwas sicherere Variante dieser Funktion ist `strncpy`. Wenn die zu kopierende Zeichenkette zu lang ist, fehlt der Kopie aber die abschließende Null.

Diese Funktionen sind ein Weg, eine Zeichenkette einer anderen zuzuweisen. Mit `a = b` geht das in C nicht, weil man nicht komplett ein Array einem anderen zuweisen kann.

2.4 Aneinanderhängen

`strcat(a, b);` kopiert die Zeichenkette `b` hinter die Zeichenkette `a`, so dass beide hintereinanderhängen. Es wird nicht geprüft, ob `a` dafür noch Platz hat, was gefährlich ist. Eine sicherere Variante dieser Funktion ist `strncat`, der man eine Maximalzahl anzuhängener Zeichen mitgeben kann (nicht die Gesamtlänge!). Diese Funktion sorgt für in jedem Fall für eine abschließende Null. Diese kommt im Zweifelsfall zu der angegeben Maximalzahl anzuhängender Zeichen hinzu.

Was ist hieran auf verborgene Weise falsch?

```
char a[] = "Donner";
char b[] = "wetter";
strcat(a, b); // Verborgener Fehler!
```

In vielen anderen Sprachen kann man `a + b` schreiben, um zwei Zeichenketten aneinanderzuhängen.

3 Ausgabe und Umwandlung

Die übliche Funktion, Zeichenketten auszugeben, ist `puts`, analog zur Ausgabe einzelner Zeichen mit `putchar`. Beides ist wesentlich schlanker als die übliche Funktion zur Ausgabe in C: `printf`. Damit diese Funktion überhaupt in den Programmspeicher des MSP430G2231 passt, muss man sie zurechtstutzen: Options > General Options > Library Options > Printf formatter: Tiny.

`printf` nimmt als erstes Argument eine Formatbeschreibung – deshalb das „f“ im Namen – und dann beliebig viele Argumente, die in die Platzhalter des Formats eingesetzt werden sollen. Die Zahl der Argumente ist in dieser Funktion also nicht fest! Beispiel:[6]

Hier sind einige Platzhalter:

- `%d` dezimale Ausgabe eines `int`
- `%x` hexadezimale Ausgabe eines `unsigned int`
- `%c` Ausgabe eines Zeichens `char`
- `%s` Ausgabe einer C-Zeichenkette
- `%%` Ausgabe des Prozentzeichens

In der „Tiny“-Variante ist die `printf`-Funktion so eigenschränkt, dass man kaum mehr tun kann. Eigentlich gehen sonst auch `long` und `double`. Ebenso kann man mit weiteren Buchstaben und Zahlen nach dem Prozentzeichen zum Beispiel einstellen, wie weit die Ausgabe mit Leerzeichen links aufgefüllt werden soll, um immer eine Mindestbreite zu haben.

Die Zahl `1234` ist etwas Anderes als die Zeichenkette `"1234"`: Die Zahl passt in eine `int`-Variable, die Zeichenkette verlangt dagegen 7 Bytes. Mit der Zahl kann man Summen usw. bilden, mit der Zeichenkette nicht. Also sind Funktionen nötig, um Zahlen in Zeichenketten zu verwandeln und umgekehrt.

In `<stdlib.h>` ist dazu zum Beispiel `atoi` deklariert („ASCII to Integer“). Diese Funktion nimmt eine Zeichenkette und versucht, am Anfang davon eine ganze Zahl zu erkennen und die als Ergebnis zurückzuliefern. Wenn das nicht gelingt, wird als Ergebnis null zurückgeliefert. So kann man allerdings das Fehlschlagen nicht von der Eingabe einer Null unterscheiden. Diese Funktion gilt deshalb als überholt.

Die umgekehrte Funktion `itoa` („Integer to ASCII“) ist nicht Teil des Standards, auch wenn sie auf einigen System existiert. Mit Hilfe der Aufgaben von Seminarzettel 6 kann man `itoa` leicht selbst entwickeln.

Eine schwergewichtige Alternative ist, `printf` in eine Zeichenkette umzuleiten. Das macht die ebenfalls in `<stdio.h>` deklarierte Funktion `sprintf`. Ihr erstes Argument ist die Zeichenkette (später genauer: die Adresse der Zeichenkette), in die „gedruckt“ werden soll. Diese Funktion prüft allerdings wieder einmal nicht, ob die angegebene Zeichenkette genug Platz hat.

Beispiel für die Verwendung:

```
char a[16];
int x = 234;
sprintf(a, "Wert: %d", x);
```

Vorsicht – auch allgemein bei den Funktionen, die mit Zeichenketten umgehen: Es muss in der Zeichenkette nicht nur Platz für die eigentlichen Zeichen sein, sondern auch für die abschließende Null.