4.4.V Datenstrukturen/Strings – Versuch

4.4.V.1 Warum Texte verarbeiten? Es gibt doch W*rd!

In der Programmierung hat man oft mit kleinen Textstücken zu tun, die man *ausgeben* möchte. Ein Beispiel ist die Anweisung:

```
printf("hello, world!\n");
```

Hier ist der Teil `hello, world!\n` so ein Textstück.

Geht man weiter in der Programmierung, so möchte man manchmal, dass ein Textstück *eingegeben* werden kann, zum Beispiel für die Anrede eines Briefes. Dieses Textstück soll dann später wieder ausgegeben werden. Dazu muss man es in einer Variablen *speichern* können.

Und dann gibt es noch den Fall, dass man ein Textstück *bearbeiten* möchte. So möchte man vielleicht alle Zeichen einer Überschrift in Großschreibung umwandeln.

Wenn so ein Textstück in einer Variablen liegt, nennt man es *Zeichenkette* oder auch *String*. Beide Begriffe meinen dasselbe.

a) Weiter zum Quiz (Seite 2)

b) Zum Beginn des Themas (Seite 1)

Quiz

Wie bezeichnet man ein kurzes Textstück in einer Variablen?

- Textvariable (Seite 3)
- Chunk (Seite 4)
- Word (Seite 4)
- String (Seite 5)
- Zeichenkette (Seite 5)

4.4.V.2 Antwort: Textvariable

Leider falsch!

Ihre Antwort „Textvariable“ war nachvollziehbar. Aber in der Praxis nennt man so eine Variable anders.

a) Zurück zum Quiz (Seite 2)

b) Zurück zum Thema (Seite 1)

4.4.V.3 Antwort: Chunk

Leider falsch!

Ihre Antwort „Chunk“ (=Happen) war nachvollziehbar. Aber in der Praxis nennt man so eine Variable anders. Nur wie?

a) Zurück zum Quiz (Seite 2)

b) Zurück zum Thema (Seite 1)

4.4.V.4 Antwort: String oder Zeichenkette

Genau richtig!

Ja, solch eine Variable nennt man „Zeichenkette“ oder (international) „String“. Weiter so!

a) Weiter zum nächsten Thema (Seite 6)

b) Zurück zum alten Thema (Seite 1)

4.4.V.5 Datentyp für Zeichenketten (Strings)

In manchen Programmiersprachen gibt es für eine Zeichenkette einen eigenen Datentyp. Beispiel: In Java schreibt man:

```
1   String s;
```

Und schon kann man `s` als eine Zeichenkette benutzen.

In C dagegen gibt es *keinen* eigenen Datentyp für eine Zeichenkette. Stattdessen benutzt man zwei grundlegende Elemente von C und kombiniert sie:

- den Datentyp char (kleine Ganzzahl)
- die Datenstruktur Array

Eine Zeichenkette erhält man als ein *Array von Elementen des Typs* `char`:

```
1   char s[10];
```

Die Zeichenkette `s` ist dann ein Array aus 10 Elementen des Typs `char`. Jedes dieser Elemente enthält genau ein Zeichen, also einen Buchstaben, eine Ziffer, ein Satz- oder ein Sonderzeichen.

Zum Beispiel könnte `s` das Wort Hallo aufnehmen (Abbildung 1).

Index	0	1	2	3	4	5	6	7	8	9
Inhalt	H	a	l	l	o					

Tabelle 1: Der String `s` enthält das Wort Hallo

a) Weiter zum Quiz (Seite 7)

b) Zum Beginn des Themas (Seite 6)

Quiz

In welchem Datentyp wird in C eine Zeichenkette gespeichert?

- string (Seite 8)
- Array von char (Seite 9)
- char (Seite 10)

4.4.V.6 Antwort: string

Leider falsch!

In anderen Programmiersprachen wie z. B. Java gibt es wirklich einen solchen Datentyp. Aber in C ist es anders.

a) Zurück zum Quiz (Seite 7)

b) Zurück zum Thema (Seite 6)

4.4.V.7 Antwort: Array von char

Genau richtig!

Es stimmt; für einen String muss man in C ein ganzes Array von `char`-Elementen anlegen. Der Vorteil ist aber: Man kann jedes Element einzeln lesen und ändern. Das ist in manchen anderen Sprachen nicht möglich.

a) Weiter zum nächsten Thema (Seite 11)

b) Zurück zum alten Thema (Seite 6)

4.4.V.8 Antwort: char

Leider falsch!

Nein, eine char-Variable ist nur eine kleine Zahl, gerade groß genug, um ein ASCII-Zeichen (0…127) unterzubringen. Merke: In eine `char`-Variable passt nur ein einziges Zeichen!

a) Zurück zum Quiz (Seite 7)

b) Zurück zum Thema (Seite 6)

4.4.V.9 Speichern und Lesen der Zeichen eines Strings

In C wird eine Zeichenkette also in einem Array von `char`-Elementen gespeichert. Das ist wie gesagt ein großer Vorteil: Man kann jedes Element einzeln ändern.

```
// Speichern von Zeichen in der Variable:
   char s[10];
   s[0] = 'H'; // speichere ein 'H' an Position 0
   s[1] = 'a'; // speichere ein 'a' an Position 1
   s[2] = 'l'; // usw.
   s[3] = 'l';
   s[4] = 'o';
```

Vom Datentyp `char` wissen wir, dass er eine kleine Zahl (meistens acht Bit) bezeichnet. Deshalb kann man den Text auch so schreiben:

```
// Speichern von Zeichen in der Variable:
   char s[10];
   s[0] =  72; // speichere ein 'H' an Position 0
   s[1] =  97; // speichere ein 'a' an Position 1
   s[2] = 108; // usw.
   s[3] = 108
   s[4] = 111;
```

Ebenso kann man auch ein Element lesen:

```
// Lesen von Zeichen aus der Variable:
   char s[10];
   char c;
   c = s[3]; // c enthaelt jetzt eine kleine Zahl
```

Das (Lesen eines Elementes) kann man zum Beispiel für eine Ausgabe benutzen:

```
putchar(s[0]); // schreibe 'h'
putchar(s[1]); // schreibe 'a'
putchar(s[2]); // schreibe 'l'
putchar(s[3]); // schreibe 'l'
putchar(s[4]); // schreibe 'o'
putchar('\n'); // schreibe neue Zeile
```

Mit einer Schleife spart man sich Programmierarbeit:

```
for(int i=0; i<5; ++i)
{
   putchar(s[i]);
}
putchar('\n');
```

a) Weiter zum Quiz (Seite 12)

b) Zum Beginn des Themas (Seite 11)

Quiz

Mit welcher Anweisung schreibt man den Kleinbuchstaben `a` an das vorderste Element der Zeichenkette `str`?

- `str[0]="a";` (Seite 13)
- `str[0]='a';` (Seite 14)
- `str[1]="a";` (Seite 15)
- `str[1]='a';` (Seite 16)
- `str='a';` (Seite 17)
- `str="a";` (Seite 18)

4.4.V.10 Antwort: `str[0]="a";`

Leider falsch!

Ein einzelnes Zeichen wird durch Hochkomma 'a' eingegrenzt, nicht durch Anführungszeichen.

a) Zurück zum Quiz (Seite 12)

b) Zurück zum Thema (Seite 11)

4.4.V.11 Antwort: `str[0]='a';`

Genau richtig!
So ist es. Alles verstanden!

a) Weiter zum nächsten Thema (Seite 19)

b) Zurück zum alten Thema (Seite 11)

4.4.V.12 Antwort: `str[1]="a";`

Leider falsch!

Ein einzelnes Zeichen wird durch Hochkomma 'a' eingegrenzt, nicht durch Anführungszeichen. Außerdem fängt eine Zeichenkette (wie jedes Array in C) mit dem Index null an.

a) Zurück zum Quiz (Seite 12)

b) Zurück zum Thema (Seite 11)

4.4.V.13 Antwort: `str[1]='a';`

Leider falsch!

In C fängt eine Zeichenkette wie jedes andere Array mit dem Index null an.

a) Zurück zum Quiz (Seite 12)

b) Zurück zum Thema (Seite 11)

4.4.V.14 Antwort: `str='a';`

Leider falsch!

Man kann nur in ein einzelnes Element schreiben. In welches soll geschrieben werden?

a) Zurück zum Quiz (Seite 12)

b) Zurück zum Thema (Seite 11)

4.4.V.15 Antwort: `str="a";`

Leider falsch!

Man kann nur in ein einzelnes Element schreiben. In welches soll geschrieben werden? Außerdem: Ein einzelnes Zeichen wird in C durch Hochkomma 'a' eingegrenzt, nicht durch Anführungszeichen.

a) Zurück zum Quiz (Seite 12)

b) Zurück zum Thema (Seite 11)

4.4.V.16 Terminierung einer Zeichenkette

Variablen sind dafür gedacht, unterschiedliche Inhalte zu besitzen. Zum Beispiel kann eine Variable `vorname` mal den einen, mal den anderen Inhalt haben.

Nun gibt es einen Unterschied zwischen den bisher bekannten Datentypen und Zeichenketten: Bei den bisherigen Datentypen war es stets so, dass jedes Bit der Variable ausgenutzt wurde: Bei einer ganzen Zahl vom Typ `int` zählt jedes Bit, ganz egal, ob die Zahl klein ist (15) oder groß (15000).

Bei Zeichenketten ist das anders. So kann die Variable `vorname` unterschiedlich gefüllt sein:

- Kurzer Vorname: `Ina` füllt nur drei Zeichen
- Langer Vorname: `Isabell-Dorothea` füllt 16 Zeichen

Die Array-Variable `vorname` muss nun so lang sein, dass jeder mögliche Inhalt hineinpasst:

```
1 char vorname[16]; // oder laenger
2 vorname[0]='I'; //  = 74
3 vorname[1]='n'; //  =110
4 vorname[2]='a'; //  = 97
```

Wenn man jetzt aber wie hier einen kurzen Vornamen hineinschreibt, was passiert dann mit den restlichen Zeichen? Und woher wissen Funktionsbausteine wie printf, wie viele Zeichen der aktuelle Vorname hat?

Dazu benutzt C einen Trick: Hinter den letzten Buchstaben des Vornamens wird ein Zeichen geschrieben, das in normalen Texten nicht vorkommt. Dieses Zeichen bedeutet, dass hier der Vorname zu Ende ist. Man nennt es den Terminator. Alle Inhalte hinter dem Terminator sind (fast immer) egal.

Und die Wahl dieses Terminators ist sehr klug: Man hat für diesen Zweck das ASCII-Zeichen mit der Nummer null genommen – genau das Zeichen, das bei der Initialisierung automatisch eingesetzt wird, falls nicht alle Elemente initialisiert wurden. Wie wird nun also `vorname` terminiert?

```
1 char vorname[16]; // oder laenger
2 vorname[0]=  74; // 'I'
3 vorname[1]= 110; // 'n'
4 vorname[2]=  97; // 'a'
5 vorname[3]=   0; // Terminator als Zahl
```

Für den Terminator gibt es auch eine andere Schreibweise mit Hochkomma:

```
1 char vorname[16]; // oder laenger
2 vorname[0]= 'I';
3 vorname[1]= 'n';
4 vorname[2]= 'a';
5 vorname[3]= '\0'; // Terminator als Ersatzdarstellung
```

Der Terminator ist *nicht* gleich dem Ziffernzeichen `'0'`, denn das hätte den Wert 48, nicht 0.

Eine Zeichenkette wird nun also durch einen Terminator `'\0'` abgeschlossen. Was sind die Konsequenzen daraus?

- Jede Zeichenkette muss terminiert werden. Andernfalls weiß `printf` nicht, wie viele Zeichen es ausgeben soll.
- Der Terminator braucht Platz.

Wir müssen also die Vereinbarung von `vorname` ändern:

```
1 char vorname[17];
```

Jetzt passt auch der oben genannte lange Vorname hinein.

`printf` liest also eine Zeichenkette bis zum Terminator `'\0'` und gibt im ersten Beispiel nur die drei Zeichen bis zum 'a' aus. Genauso arbeiten viele andere Funktionsbausteine aus der C-Standardbibliothek: Sie arbeiten nur bis zum Terminator. Und viele Bausteine, die Zeichenketten schreiben, hängen an die selbstgebaute Zeichenkette einen Terminator an.

Merke: Ein Terminator muss immer am Ende der Zeichenkette stehen – nach dem letzten aktuell gültigen Zeichen. Er muss dabei aber immer *im* Array stehen – genauso, wie man einen Grenzzaun immer auf dem eigenen Grundstück bauen sollte und nicht auf dem des Nachbarn.

a) Weiter zum Quiz (Seite 21)

b) Zum Beginn des Themas (Seite 19)

Quiz

Wie viele Elemente braucht man mindestens für eine Zeichenkette, die den Text `"welt"` speichern kann?

- 4 (Seite 22)
- 5 (Seite 23)
- 6 (Seite 24)

4.4.V.17 Antwort: 4

Leider falsch!

Es fehlt noch ein Byte für den Terminator.

a) Zurück zum Quiz (Seite 21)

b) Zurück zum Thema (Seite 19)

4.4.V.18 Antwort: 5

Genau richtig!

Sie haben Platz für den Terminator eingeräumt. Behalten Sie das im Kopf!

a) Weiter zum nächsten Thema (Seite 25)

b) Zurück zum alten Thema (Seite 19)

4.4.V.19 Antwort: 6

Leider falsch!

So viel Platz ist auch nicht nötig, schadet aber meistens nicht.

a) Zurück zum Quiz (Seite 21)

b) Zurück zum Thema (Seite 19)

4.4.V.20 Initialisierung einer Zeichenkette

Eine Zeichenkette kann – wie jedes Array – bei der Vereinbarung mit einem bestimmten Inhalt initialisiert (=vorbelegt) werden:

```
1    char zk[10]={'H', 'a', 'l', 'l', 'o'};
```

In diesem Beispiel sind die ersten fünf Bytes wie gewünscht vorbelegt. Was passiert mit den anderen fünf Bytes? Bei einem Array gilt die Regel, dass bei einer Initialisierung alle nicht explizit initialisierten Bytes mit null vorbelegt werden. Also sind die anderen fünf Bytes mit null vorbelegt. Das ist bei einer Zeichenkette sehr günstig: Die null ist gleichzeitig der Terminator. Also besteht der Inhalt der Variablen aus den fünf explizit festgelegten Bytes und fünf Terminatoren (Abbildung 2).

Index	0	1	2	3	4	5	6	7	8	9
Inhalt als Zeichen	H	a	l	l	o	\0	\0	\0	\0	\0
Inhalt als Zahl	72	97	108	108	111	0	0	0	0	0

Tabelle 2: Der String s enthält das Wort Hallo und fünf Terminatoren

Man kann die Zeichenkette auch in einer Kurzschreibweise initialisieren; das Ergebnis ist identisch:

```
1    char zk[10]="Hallo";
```

Wenn man dem Array zu wenige Elemente gibt, dann ist leider kein Platz mehr für einen Terminator. Im folgenden Beispiel ist das in Zeile 5 für die Variable t der Fall:

termfehlt.c

```
1 #include <stdio.h>
2 int main(void)
3 {
4     char s[5]="FORD";
5     char t[4]="OPEL";
6     printf("%s\n", t);
7     return 0;
8 }
```

Das Ergebnis ist:

Terminal

```
schueler@debian964:~$ gcc termfehlt.c
schueler@debian964:~$ a.out
OPELFORD
```

Man sieht: Die Variable t enthält keinen Terminator; also liest der Funktionsbaustein printf über das Ende von t hinaus; hinter dem Ende von t steht zufällig der Inhalt von s, so dass der auch gleich mit ausgegeben wird. Bei s dagegen wurde Platz gelassen für einen Terminator, deshalb hört printf am Ende von s auf mit der Ausgabe von Zeichen.

a) Weiter zum Quiz (Seite 26)

b) Zum Beginn des Themas (Seite 25)

Quiz

Wie viele Terminatoren werden bei dieser Variablen gesetzt: `char u[7]="welt";`?

- keiner (Seite 27)
- 1 (Seite 28)
- 3 (Seite 29)

4.4.V.21 Antwort: keiner

Leider falsch!

Das stimmt nicht: Bei der Initialisierung eines Arrays wird jedes nicht explizit initialisierte Element mit 0 bzw. `'\0'` vorbelegt. Vier von sieben Bytes sind explizit vorbelegt, bleiben also drei Null-Bytes übrig.

a) Zurück zum Quiz (Seite 26)

b) Zurück zum Thema (Seite 25)

4.4.V.22 Antwort: 1

Leider falsch!

Es stimmt schon: Es wird ein Terminator eingefügt – aber nicht, weil das hier eine Zeichenkette ist, sondern weil bei der Initialisierung eines Arrays *jedes* Element, das nicht extra genannt wurde, mit einem Null-Byte belegt wird – und dieses Null-Byte gilt hier bei einer Zeichenkette als Terminator. Praktisch, oder?

a) Zurück zum Quiz (Seite 26)

b) Zurück zum Thema (Seite 25)

4.4.V.23 Antwort: 3

Genau richtig!

Genau richtig, Sie haben die Sache mit der Initialisierung verstanden!

a) Weiter zum nächsten Thema (Seite 30)

b) Zurück zum alten Thema (Seite 25)

4.4.V.24 So viel Platz wie nötig: Den Compiler zählen lassen

Manchmal möchte man, dass ein String genau so viele Zeichen bekommt, dass er bei einer bestimmten Initialisierung von der Länge her passt. Dann darf man bei der Vereinbarung die Längenangabe weglassen (wie bei jedem Array):

hans.c

```
1 #include <stdio.h>
2 int main(void)
3 {
4     char a[]="Hans";
5     printf("a: %i Bytes\n", sizeof a);
6     return 0;
7 }
```

Jetzt fragt man sich: Ist die Größe so gewählt, dass ein Terminator Platz hat?

Terminal

```
schueler@debian964:~$ gcc hans.c
schueler@debian964:~$ a.out
a: 5 Bytes
```

Offenbar ja: Man muss keinen Extra-Platz reservieren; es wird ein Byte für den Terminator gelassen, und dieses Bytes wird mit dem Terminator `'\0'` vorbelegt.

Und nun der Vollständigkeit halber noch einmal für die Lang-Schreibweise der Initialisierung:

emil.c

```
1 #include <stdio.h>
2 int main(void)
3 {
4     char b[]={'E','m','i','l'};
5     printf("b: %i Bytes\n", sizeof b);
6     printf("b: %s\n", b);
7     return 0;
8 }
```

Hat auch hier der Terminator Platz?

Terminal

```
schueler@debian964:~$ gcc emil.c
schueler@debian964:~$ a.out
a: 4 Bytes
```

Hier ist also ein Unterschied zwischen den beiden Schreibweisen: In der Langschreibweise wird kein Byte für den Terminator reserviert und es wird auch kein Null-Byte gesetzt. Hier findet *keine* Terminierung statt.

Das kann man auch an folgendem Beispiel sehen:

termfehlt2.c

```
1 #include <stdio.h>
2 int main(void)
3 {
4     char a[]="Hans";
5     char b[]={'E','m','i','l'};
6     printf("a: %i Bytes\n", sizeof a);
7     printf("b: %i Bytes\n", sizeof b);
```

```
8     printf("a: %s\n", a);
9     printf("b: %s\n", b);
10    return 0;
11 }
```

Terminal

```
schueler@debian964:~$ gcc termfehlt2.c
schueler@debian964:~$ a.out
a: 5 Bytes
b: 4 Bytes
a: Hans
b: EmilHans
```

Wozu ist dann die Lang-Schreibweise gut? Man kann sie immer dann einsetzen, wenn man in einem Datenfeld keinen Platz für einen Terminator hat und deshalb auch keinen einsetzen lassen möchte.

a) Weiter zum Quiz (Seite 32)

b) Zum Beginn des Themas (Seite 30)

Quiz

Wie viele Bytes hat das folgende Array: `char stadt[]="Melle";`?

- fünf (Seite 33)
- sechs (Seite 34)
- undefiniert (Seite 35)

4.4.V.25 Antwort: fünf

Leider falsch!

Das stimmt nicht: Bei der üblichen Kurzform der Initialisierung einer Zeichenkette wird ein Byte für den Terminator eingefügt und mit '\0' vorbelegt.

a) Zurück zum Quiz (Seite 32)

b) Zurück zum Thema (Seite 30)

4.4.V.26 Antwort: sechs

Genau richtig!

Das ist richtig: Es wird genau ein Byte für den Terminator eingefügt. Und das wird mit dem Null-Byte '\0' vorbelegt.

a) Weiter zum nächsten Thema (Seite 36)

b) Zurück zum alten Thema (Seite 30)

4.4.V.27 Antwort: undefiniert

Leider falsch!

Ein Array mit undefinierter Länge gibt es in C nicht. Und das braucht man hier auch gar nicht: Die Zeichen des Initialisierer-Strings werden abgezählt; zusätzlich wird bei dieser Art der Initialisierung ein Byte für den Terminator hinzugefügt.

a) Zurück zum Quiz (Seite 32)

b) Zurück zum Thema (Seite 30)

4.4.V.28 Ausgabe einer Zeichenkette

– wird fortgesetzt –