Darstellung von Strings

Eine der wichtigsten Anwendungen von Arrays ist die Darstellung und Verarbeitung von **Strings** (**Zeichenfolgen**). Bisher haben wir schon regen Gebrauch davon gemacht, und zwar haben wir es mit der konstanten Form der Strings zu tun gehabt, den Stringkonstanten. Stringkonstanten werden immer in doppelten Anführungsstrichen eingeschlossen. Sie haben bei jeder Bildschirmausgabe mit Stringkonstanten gearbeitet.

z.B. `cout << "Hello World";`

In diesem Kapitel wollen wir uns damit beschäftigen wie man Strings in Variablen speichert.

Wenn Sie sich einen String genauer ansehen, stellen Sie fest, daß es sich dabei um eine geordnete Folge von Zeichen handelt. Hätten wir eine geordnete Folge von **`int-`** Werten, würden wir diese in einem **`int-`** Feld speichern.
Wenn wir das nun auf die Strings übertragen, dann speichern wir eine Folge von Zeichen am besten in einem **`char-`** Feld.

Strings sind nichts anderes als ein Variablenfeld vom Typ **`char`**.

Beispiel:

```
char a[80];
```

Es wird ein Feld definiert, das 80 Zeichen speichern kann.

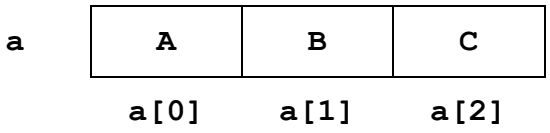
Endekennung

Eine Stringkonstante wird intern als **char-** Feld dargestellt, dessen Länge um eins größer ist als die Anzahl der Zeichen in der Stringkonstanten. Als letztes Zeichen wird automatisch eine Endekennung eingefügt. Die Endekennung eines Strings hat in C++ den Wert 0. Fehlt das Null- Zeichen hat man nur einzelne Zeichen in einem **char-** Array, jedoch noch keinen String.

Beispiel:

```
char a[3];
for (int i=0, i<3; i++)
a[i] = 'A' +i;
```

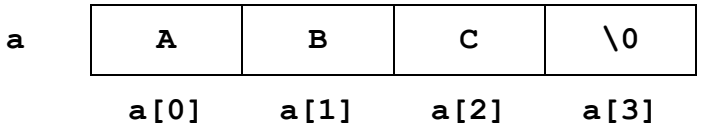
Das Array enthält demnach keinen String, sondern lediglich die Zeichen 'A', 'B' und 'C'.



Im Gegensatz dazu würde das Array a nach

```
char a[4];
for (int i=0, i<3; i++)
a[i] = 'A' +i;
a[3] = '\0';
```

einen String, nämlich die Zeichenkette "ABC" speichern.



Wegen des zusätzlichen Null- Zeichens belegt ein String stets ein Byte mehr Platz im Speicher, als die eigentliche Zeichenfolge Zeichen hat. Das muß natürlich bei der Angabe der Elementezahl eines Arrays berücksichtigt werden.

Der "**Hello World**"- String müßte also als 12 Zeichen langes **char-** Feld gespeichert werden.

Beispiel:

H	e	l	l	o		W	o	r	l	d	\0
a[0]	a[1]	a[2]	a[39	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]

Ein **char-** Array, das den String "Hello World" enthält

Selbstverständlich läßt sich die Zeichenfolge "Hello World" auch in einem 15elementigen Array speichern. Es müssen nicht genau n+1 Elemente existieren, sondern das Array muß über mindestens n+1 Elemente verfügen.

H	e	l	l	o		W	o	r	l	d	\0			
a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]	a[11]	a[12]	a[13]	a[14]

Nur die ersten 12 Elemente sind belegt, die restlichen bleiben unbenutzt bzw. haben zufällige Inhalte, die nichts mit dem String zu tun haben. Das Null- Zeichen sorgt dafür, daß bei einer Ausgabe nur die Inhalte a[0], ... a[11] angezeigt werden. Sollte das Array mehrere Null-Zeichen enthalten, so ist das erste (von links gesehen) als Endekennung maßgeblich.

Regel:

Ein String darf mit Endekennung nur gleich oder kleiner als das ihn beherbergende Feld sein, niemals größer.

Ein- und Ausgabe von Strings

Aufgrund des besonderen Status eines **char-** Arrays wird es von den Ein- / Ausgabefunktionen besonders unterstützt, z.B. von **cout**.

Beispiel:

```
#include <iostream>
using namespace std;
int main ()
{
char a[7];
     a[0] = 'S';
     a[1] = 't';
     a[2] = 'r';
     a[3] = 'i';
     a[4] = 'n';
     a[5] = 'g';
     a[6] = '\0';
     cout << a << endl;[1]
}
```

Natürlich werden die Strings auch von **cin** unterstützt, so daß man auch Zeichenfolgen einlesen kann. Hier gibt es jetzt allerdings ein kleines Problem.
Sollten Sie eine Zeichenfolge eingeben die ein Leerzeichen enthält, wird später bei der **cout-** Anweisung nur der Text ausgegeben der vor dem ersten Leerzeichen steht. Doch dazu kommen wir etwas später.

[1] Zur Erinnerung: Der Name eines Arrays ohne eckige Klammern steht für die Startadresse des Feldes.

Beispiel:

```
#include <iostream>
using namespace std;
int main()
{
     char a[81];
     cout << "Bitte einen max. 80 Zeichen langen Text
     eingeben:" << endl;
     cin >> a;
     cout << "Sie gaben \"" << a << "\" ein." << endl;
     return (0);
}
```

getline

Mit der einfachen **cin-** Anweisung läßt sich das Problem der Leerzeichen bei der Texteingabe nicht beheben. Von **cin** wird jedoch speziell für Strings eine Eingabefunktion zur Verfügung gestellt. Sie heißt **getline** und hat folgenden Funktionskopf:

```
cin.getline(char *string, int groesse, char
trennzeichen='\n');
```

Der Funktion muß die Adresse des zu beschreibenden Strings und dessen Länge übergeben werden. Das Trennzeichen wurde mit einem **default-** Parameter versehen, und braucht deshalb beim Aufruf nur dann angegeben werden, wenn ein anderes gewünscht wird. Der eingelesene String wird automatisch mit einer Endekennung versehen.

Beispiel:

```
#include <iostream>
using namespace std;
int main()
{
char a[81];
     cout << "Bitte einen max. 80 Zeichen langen Text
     eingeben:" << endl;
     cin.getline(a,80);
     cout <<"Sie gaben \""<< a << "\" ein." << endl;
}
```

Denken Sie daran, daß die Länge des **char-** Feldes und die Länge des in ihm enthaltenen Strings nicht identisch sein müssen.

Die cstring- Funktionen

Zur Manipulation von Strings stehen Funktionen aus der Standardbibliothek zur Verfügung. Die Funktionen werden mit **cstring** eingebunden:

```
#include <cstring>
```

Die wichtigsten Funktionen von cstring[1]

memchar	Zeichen im Speicherblock suchen
memcmp	Speicherblöcke vergleichen
memcpy	Speicherblöcke kopieren
memmove	sicheres kopieren von Speicherblöcken
memset	Speicherblock initialisieren
strcat	String an einen anderen hängen
strchr	Zeichen im String suchen
strcmp	zwei Strings vergleichen
strcoll	zwei Strings umgebungsabhängig vergleichen
strcpy	String kopieren
strerror	Umwandlung eines Fehlers in verbale Form
strlen	Länge des Strings ermitteln
strncat	Teil eines Strings an einen anderen hängen
strncmp	Teile von zwei Strings vergleichen
strncpy	Teile eines Strings kopieren
strrchr	Zeichen vom Stringende aus suchen
strstr	prüft daraufhin, ob String Teil eines anderen ist

Initialisierung von Strings

Eines von mehreren Verfahren, eine Stringvariable zu erzeugen, besteht darin, ein **`char`**-Array mit einer Zeichenkette zu initialisieren.

Beispiel:

```
char a[7] = { 'S', 't', 'r', 'i', 'n', 'g', '\0'};
```

oder

```
char a[ ] = { 'S', 't', 'r', 'i', 'n', 'g', '\0'};
```

[1] Eine ausführliche Erklärung der Funktionen finden Sie am Ende des Kapitels

Allerdings ist diese Art etwas mühsam. C++ erlaubt uns eine abkürzende Schreibweise, die statt der Liste von Einzelwerten eine entsprechende Stringkonstante als Initialisierungswert erlaubt.

Beispiel:

```
char a[7] = "String";
```

oder

```
char a[ ] = "String";
```

In beiden Fällen wird der String a mit "String" initialisiert. Eine Zuweisung im Programm wie z.B.

```
a = "Name";
```

ist nicht erlaubt, weil eine Stringkonstante immer für ihre Adresse steht.[1]

Regel:

Eine Stringkonstante steht immer für ihre Adresse.

Für die Zuweisung muß man sich mit einem Trick behelfen. Wollen Sie einem String einen neuen Inhalt zuweisen, müssen Sie auf die Funktion **`strcpy`** zurückgreifen.
Syntax:

```
strcpy (ziel, quelle);
```

`strcpy` kopiert den Inhalt des Strings **`quelle`** in den String **`ziel`**, inclusive Endekennung. Der Funktion werden die Adressen der Strings übergeben.
Beispiel:

```
strcpy (a, "Name")
```

[1] Wäre a ein Zeiger auf char, wäre die Schreibweise richtig. a hätte als Adresse dann den Anfang der Stringkonstanten. Auf diese Weise wäre es möglich während der Laufzeit Änderungen an der Stringkonstanten vorzunehmen, was jedoch unsaubere Programmierung wäre.

Dieser Aufruf ist korrekt. Der Name eines Strings ohne eckige Klammern steht für seine Adresse, eine Stringkonstante repräsentiert ebenfalls ihre Adresse. Stringzuweisungen außerhalb einer Definition bedürfen immer der **strcpy-** oder einer ähnlichen Funktion.

Strings als Funktionsparameter

Strings werden als Funktionsparameter genauso behandelt wie andere Variablenfelder. Es wird die Adresse des Strings an die Funktion übergeben. Die Funktion muß in ihrem Funktionskopf einen Zeiger auf **char** deklarieren.
Beispiel:

```
#include <iostream>
#include <cstring>
using namespace std;
void fkt (char *a)
{
     cout << "In der Funktion  :" << a << endl;
     strcpy (a, "Anton");
     cout << "Nach strcpy      :" << a << endl;
}

int main ()
{
char x[15] = "Willibald";

     cout << "Vor der Funktion   :" << x << endl;
     fkt (x);
     cout << "Nach der Funktion  :" << x << endl;
     return (0);
}
```

Wird der String innerhalb der Funktion verändert, hat es Auswirkungen auf den String der aufrufenden Funktion, weil die Adresse übergeben wurde und keine Kopie. Soll das Original

unverändert bleiben, müssen Sie eigenhändig eine Kopie anfertigen, mit der dann gearbeitet wird. Mit folgender Funktion bleibt der Originalstring unverändert.

Beispiel:

```
void fkt (char *a)
{
char b[15];
strcpy (b, a);
cout << "In der Funktion  :" << b << endl;
strcpy (b, "Anton");
cout << "Nach strcpy      :" << b << endl;
}
```

Zeichenweiser Zugriff auf Strings

Es gibt die Möglichkeit auf die einzelnen Zeichen eines Strings zuzugreifen
Beispiel:

```
a = 0;
while (x[a])
cout << x[a++];
```

Die **while-** Schleife bricht bei der Endekennung ab. Wenn man einen Zeiger auf einen String zur Verfügung hat, gibt es eine zweite Variante.
Beispiel:

```
char *z;
z = x;
while (*z)
cout << *z++;
```

Hier wird ein Zeiger auf **char** definiert, dem die Adresse von **x** zugewiesen wird. Durch die Dereferenzierung wird auf das Zeichen an der Adresse zugegriffen, die der Zeiger gespeichert hat.

Stringfelder

Es können mehrere Strings zu einem Feld zusammengefaßt werden. Das entspricht dann einem zweidimensionalen **char-** Feld. z.B.

```
char x[5] [10];
```

Diese Anweisung definiert 5 Strings mit einem Fassungsvermögen von 10 Zeichen. Es ist auch eine gleichzeitige Initialisierung möglich:

```
char x[5] [10]= {"Hansi","Hugo","Otto",Frank","Herbert"};
```

Auf die einzelnen Strings im Feld kann sehr einfach zugegriffen werden. Mit der folgenden Schleife werden alle fünf Namen auf den Bildschirm gebracht:

```
for (a=0; a<5; a++)
     cout (x[a]);
```

Hier haben wir eine Erweiterung der Regel, daß der Name eines Strings ohne eckige Klammern die Adresse repräsentiert.

Regel:

Der Name eines Stringfeldes mit einem Index weniger als in der Definition, repräsentiert die Adresse eines speziellen Strings.

In der **for-** Schleife wurde dieses angewendet, da **x[a]** einen Index weniger hat als in seiner Definition.

cstring- Funktionen

memchar `void *memchr(const void *feldadr, int z, size_t groesse);`

sucht nach einem Feld der Größe **groese**, beginnend bei **feldadr** nach dem ersten vorkommen des Zeichens z. Bei Erfolg wird die Adresse des Zeichens, bei Mißerfolg 0 zurückgegeben.

memcmp `int memcmp(const void *feldadr1, const void *feldadr2, size_t groesse);`

vergleicht zwei Felder der Größe **groesse**, das eine beginnend bei **feldadr1**, das andere beginnend bei **feldadr2**. Bei Gleichheit wird 0 zurückgegeben. Bei Ungleichheit wird ein positiver Wert zurückgegeben, wenn das Element des ersten Feldes größer ist als das des zweiten. Umgekehrt wird ein negativer Wert zurückgegeben.

memcpy `void *memcpy(void *feldadr1, const void *feldadr2, size_t groesse);`

kopiert **groesse** Elemente des Feldes beginnend bei **feldadr2** in das Feld beginnend bei **feldadr1**. Zurückgegeben wird **feldadr1**.

memmove `void *memmove(void *feldadr1, const void *feldadr2, size_t groesse);`

ähnelt in seiner funktionsweise **memcpy**, nur daß bei einer teilweisen Überlappung der Felder ein Verlorengehen von Informationen ausgeschlossen ist.

memset `void *memset(void *feldadr, int z, size_t groesse);`

füllt alle Elemente eines bei **feldadr** beginnenden Feldes der Größe **groesse** mit dem Zeichen **z**.

strcat `char *strcat(char *ziel, const char *quelle);`

kopiert den String **quelle** mitsamt der Endekennung hinter den String **ziel**, wobei die Endekennung von **ziel** mit dem ersten Zeichen von **quelle** überschrieben wird. Es wird **ziel** zurückgegeben.

strchr `char *strchr(const char *s, char z);`

liefert die Adresse des ersten in **s** vorkommenden Zeichens **z**, wobei die Endekennung ebenfalls berücksichtigt wird. Bei Mißerfolg wird 0 zurückgegeben.

strcmp `int strcmp(const char *s1, const char *s2);`

vergleicht die beiden Strings **s1** und **s2**. Bei Gleichheit wird 0 zurückgegeben. Ist das erste unterschiedliche Zeichen bei **s1** größer als bei **s2**, wird ein positiver, im umgekehrten Fall ein negativer Wert zurückgegeben.

strcoll `int strcoll(const char *s1, const char *s2);`

hat eine funktionsweise wie **strcmp**, nur daß die Vergleichsregel der Zeichen von der Umgebung abhängig ist.

strcpy `char *strcpy(char ziel, const char *quelle);`

kopiert den String **quelle** mitsamt der Endekennung in den String **ziel**. Es wird **ziel** zurückgegeben

strerror `char *strerror(int fehlernummer);`

liefert die Adresse eines Strings, der den durch **fehlernummer** repräsentierten Fehler verbal formuliert.

strlen `size_t strlen(const char *s);`

liefert die Länge des Strings **s** ohne Endekennung zurück

strncat `char *strncat(char *ziel, const char *quelle, size_t n);`

kopiert die ersten **n** Zeichen des Strings **quelle** mit Hinzufügen einer Endekennung hinter den String **ziel**, wobei die Endekennung von **ziel** mit dem ersten Zeichen von **quelle** überschrieben wird. Es wird **ziel** zurückgegeben. Sollte **quelle** kleiner **n** Zeichen sein, werden **strlen(quelle)** Zeichen plus Endekennung kopiert.

strncmp `int strncmp(const char *s1, const char *s2, size_t n);`

vergleicht die ersten **n** Zeichen der beiden Strings **s1** und **s2**. Bei Gleichheit wird 0 zurückgegeben. Ist das erste unterschiedliche Zeichen bei **s1** größer als bei **s2**, wird ein positiver, im umgekehrten Fall ein negativer Wert zurückgegeben.

strncpy `char *strncpy(char *ziel, const char *quelle, size_t n);`

kopiert die ersten **n** Zeichen des Strings **quelle** in den String **ziel**. Es wird **ziel** zurückgegeben. Sollte **quelle** kürzer als **n** Zeichen sein, werden **strnlen(quelle)** Zeichen kopiert und die restlichen **n-strlen(quelle)** Zeichen mit 0 aufgefüllt.

strrchr `char *strrchr(const char *s, int z);`

liefert die Adresse des letzten in **s** vorkommenden Zeichens **z**, wobei die Endekennung ebenfalls berücksichtigt wird. Bei Mißerfolg wird 0 zurückgegeben.

strstr `char *strstr(const char *s, const char *suchstr);`

liefert die Adresse der ersten Zeichenfolge von **s**, welche mit **suchstr** ohne Endekennung übereinstimmt. Fall keine Zeichenfolge übereinstimmt, wird 0 zurückgegeben.

Kontrollfragen:

1. Was unterscheidet einen String von einem **`char-`** Feld?

2. Welchen Nachteil hat die Eingabe eines Strings mit **`cin`**?

Übungen:

Übung 1. (leicht)
Schreiben Sie eine Funktion namens **`ostrcpy`**, die wie **`strcpy`** den Inhalt eines String, einschließlich Endekennung in einen anderen kopiert, aber keinen Rückgabewert hat. Der Funktionsaufruf soll wie folgt lauten: **`ostrcpy (ziel,quelle)`**. Benutzen Sie nur selbstgeschriebene Funktionen.
Schreiben Sie auch eine **`main-`** Funktion, mit der Sie Ihre Funktion testen können.

Übung 2. (leicht)
Schreiben Sie eine Funktion namens **`ostrlen`**, die die Länge eines Strings einschließlich der Endekennung zurückgibt. Benutzen Sie zur Lösung nur selbstgeschriebene Funktionen.
Schreiben Sie auch eine **`main-`** Funktion, mit der Sie Ihre Funktion testen können.

Übung 3. (mittel)
Schreiben Sie eine Funktion **`upstring`**, die alle Kleinbuchstaben eines Strings in Großbuchstaben umwandelt, Großbuchstaben und andere Zeichen aber unverändert läßt. Die Funktion soll die Anzahl der umgewandelten Buchstaben zurückliefern.
z.B. "Hello World" wird "HELLO WORLD" und als Wert würde 8 zurückgeliefert werden. Schreiben Sie auch eine **`main-`** Funktion, mit der Sie die Lösung überprüfen können.
Tip: Behandeln Sie jedes Zeichen einzeln. Benutzen Sie die Funktionen **`toupper`** und **`islower`** (vgl. Seite II-87 char-Funktionen).

Übung 4. (mittel)

Schreiben Sie eine Funktion namens **`reversstring`**, die einen String umdreht. Die Funktion soll keinen Rückgabewert haben.

z.B. "Hello World" wird "dlroW olleH"

Schreiben Sie auch eine **`main-`** Funktion, um die Lösung zu testen.

Tips:

Eine simple Methode besteht darin, einen zweiten leeren String anzulegen, und den Originalstring dann von hinten zeichenweise in den leeren String zu kopieren.

Eine andere Möglichkeit wäre, jeweils das erste Zeichen mit dem letzten Zeichen, das zweite mit dem vorletzten Zeichen, usw. zu vertauschen.

Übung 5. (schwer)

Schreiben Sie eine Funktion namens **`mixstring`**, der Sie drei Strings übergeben **`(ziel, quelle1, quelle2)`**, wobei der Zielstring aus den beiden Quellstrings so erzeugt wird, daß er das erste Zeichen des ersten Strings, das erste Zeichen des zweiten Strings, das zweite Zeichen des ersten Strings, usw. enthält.

Beispiel:

Quelle1 "Anton" und Quelle2 "Willi", dann sollte die Funktion den Zielstring "AWnitlolni" erzeugen.

Sollte einer der Strings länger sein, dann sollen die übrig gebliebenen Zeichen angehängt werden.

Die Funktion liefert keinen Wert zurück. Benutzen Sie nur selbstgeschriebene Funktionen.

Schreiben Sie auch eine **`main-`** Funktion, um Ihre Funktion zu testen.

Tips:

Damit bei unterschiedlich langen Strings nicht über die Grenzen des kürzeren hinweg Zeichen kopiert werden, muß für jeden Quellstring eine extra Abfrage implementiert werden, ob das Ende schon erreicht wurde.

Übung 6. (schwer)

Schreiben Sie eine Funktion namens **`ostrstr`**, die überprüft, ob ein String in einem anderen enthalten ist. Ist das der Fall, soll **`ostrstr`** den Index zurückgeben, an dem der String steht, und nicht wie **`strstr`** die Adresse. Ist der gesuchte String nicht enthalten, soll die Funktion -1 zurückgeben.

Beispiel: String1 = "Otto", String2 = "Und Otto ging baden", dann sollte **`ostrstr(String1,String2)`** den Wert 4 zurückliefern.

Benutzen Sie nur selbstgeschriebene Funktionen.

Schreiben Sie auch eine **`main-`** Funktion, um die Lösung zu testen.

Tips:

Die Lösung besteht aus zwei verschachtelten Schleifen.

Die innere Schachtel muß den zu suchenden String zeichenweise mit einem Ausschnitt des Strings vergleichen, in dem gesucht werden soll. Die äußere Schleife sorgt dafür, daß bei keiner Übereinstimmung der zu vergleichende Ausschnitt innerhalb des Strings, in dem gesucht wird, um ein Zeichen verschoben wird.

Übung 7. (leicht)

Schreiben Sie eine Funktion namens **`leftstr`**, die Sie mit **`leftstr(ziel,quelle,anz)`** aufrufen können, und die die ersten **`anz`** Zeichen des Quellstrings in den Zielstring kopiert. Der Zielstring soll mit einer Endekennung versehen werden. Die Funktion soll keinen Rückgabeparameter haben. Die Funktion soll vor Bereichsüberschreitung geschützt werden. z.B. soll verhindert werden, daß die ersten 8 Zeichen eines nur 6 Zeichen langen Strings kopiert werden. Benutzen Sie nur selbstgeschriebene Funktionen.

Übung 8. (leicht)

Schreiben Sie eine Funktion namens **`rightstr`**, die Sie mit **`rightstr(ziel,quelle,anz)`** aufrufen können, und die die letzten **`anz`** Zeichen des Quellstrings in den Zielstring kopiert. Der Zielstring soll mit einer Endekennung versehen werden. Die Funktion soll keinen Rückgabeparameter haben. Die Funktion soll vor Bereichsüberschreitung geschützt werden. z.B. soll verhindert werden, daß die letzten 8 Zeichen eines nur 6 Zeichen langen Strings kopiert werden. Benutzen Sie nur selbstgeschriebene Funktionen.

Übung 9. (leicht)

Schreiben Sie eine Funktion namens **`midstr`**, die Sie mit **`midstr(ziel, quelle, pos, anz)`** aufrufen können und die ab dem Index **`pos`** des Quellstrings **`anz`** Zeichen in den Zielstring kopiert. Der Zielstring soll mit einer Endekennung versehen werden. Die Funktion soll keinen Rückgabewert haben. Schützen Sie die Funktion vor Bereichsüberschreitung. Benutzen Sie nur selbstgeschriebene Funktionen.

Übung 10. (schwer)

Schreiben Sie eine Funktion **`toWord`**, der Sie einen **`unsigned-long`**- Wert und die Adresse eines Strings übergeben. Die Funktion wandelt diesen Wert dann in ein Wort um und speichert dieses Wort in dem String, dessen Adresse übergeben wurde.
Beispiel:
Aus 0 wird "Null"
Aus 1 wird "Eins"
Aus 2000012 wird "Zweimillionenzwoelf"
Die größte umwandelbare Zahl sollte 999999999999 sein.
Schreiben Sie auch eine **`main`**- Funktion, um Ihre Funktion zu testen.

Tips:
Es ist wichtig, sich über die Gemeinsamkeiten der geschriebenen Zahlen im klaren zu sein. Dadurch erkennt man die Besonderheiten.
Es bietet sich an, die Gemeinsamkeiten in einzelnen Funktionen zusammenzufassen.

Lösungen:

Antwort zu Frage 1.

Strings werden von Ein-/ Ausgabefunktionen speziell unterstützt. Das letzte Feldelement eines korrekten Strings ist immer 0 (Endekennung).

Antwort zu Frage 2.

cin wertet Leerzeichen als Trennzeichen von mehreren Eingaben, man muß dann auf **getline** zurückgreifen.

Lösung zu Übung 1.

```
#include <iostream>
#include <cstring>
using namespace std;
void ostrcpy(char *a, char *b)
int z = 0;
{
do
{
     a[z] = b[z];
     z = z+1;
     }
     while (b[z-1] != 0);
}
int main()
{
     char quelle[160], ziel[160];
     cout << "Bitte String eingeben:";
     cin.getline(quelle,160);
     ostrcpy(ziel,quelle);
     cout << "Original String :\""<<quelle<<"\"."<<endl;
     cout << "Kopierter String:\""<<ziel<<"\"."<<endl;
}
```

Lösung zu Übung 2.

```
#include <iostream>
#include <cstring>

using namespace std;

int ostrlen(char *a)
{
     int z=0;
     while (a[z++]);
     return (z);
}

int main()
{
     int x=0;
     char quelle[160];

     cout << "Bitte String eingeben:";
     cin.getline(quelle,160);

     x=ostrlen(quelle);

     cout << "Original String :\"" << quelle << "\"." << endl;
     cout << "String Laenge   : " << x << " Zeichen inkl.
       Endekennung"<<endl;

     return(0);
}
```

Bei dieser Lösung wird eine Zählvariable gebraucht. Die **while-** Anweisung bricht ab, wenn die Bedingung falsch ist. Die Bedingung ist dann falsch, wenn **a[z]** gleich Null ist, was der Endekennung entspricht. Die Endekennung wird mitgezählt, weil **z** durch das Postinkrement auch dann noch einmal erhöht wird, wenn **while** durch die Endekennung abbricht.

Lösung zu Übung 3.

```
#include <iostream>
#include <cctype>

using namespace std;

int upstring (char *a)
{
     int z=0;
     while (*a)
     {
          if (islower(*a))
          {
          *a = toupper (*a);
          z++;
     }
     a++;
}

int main()
{
     char quelle[160];

     int x = 0;

     cout << "Bitte String eingeben:";
     cin.getline(quelle,160);

     x = upstring(quelle);

     cout << "Umgewandelter String:\""<<quelle<<"\"."<<endl;
     cout << "Umwandlungen :" << x << endl;
  }
```

Die Funktion macht Gebrauch vom Dereferenzierungsoperator, um eine Zählvariable zu sparen. Die **while-** Schleife wird solange ausgeführt, bis **a** auf die Endekennung zeigt. Zu Beginn liefert ***a** also das erste Zeichen. Es wird überprüft ob es sich um einen Kleinbuchstaben handelt. Wenn ja, wird das Zeichen in einen Großbuchstaben umgewandelt und die Zählvariable um eins erhöht. Schließlich wird **a** um eins erhöht, so daß der Zeiger nun auf das nächste Zeichen des Strings zeigt.

Lösung zu Übung 4.

```
#include <iostream>
#include <cstring>

void reversstring(char *s)
{
char k[1000];

     int x, y=0;
     strcpy(k,s);
     x=strlen(k)-1;
     while(x>=0)
     s[y++]=k[x—];
}

int main()
{
   char quelle[160];
   cout << "Bitte String eingeben:";
   cin.getline(quelle,160);
   reversstring(quelle);
   cout << "Umgewandelter String:\""<<quelle<<"\"."<<endl;
}
```

Hier wird zuerst eine Kopie des Strings angefertigt. **x** bekommt den Index des letzten Zeichens des Strings **k**, so daß mit **x** der String **k** von hinten nach vorne und mit **y** der String **s** von vorne nach hinten durchlaufen wird. **x** bekommt den um eins erniedrigten Wert von **strlen** zugewiesen, weil das erste Zeichen des Strings den Index 0 hat.
Diese Lösung hat den Nachteil, daß die Strings, die mit der Funktion bearbeitet werden können, begrenzt sind. Sinnvoller wäre daher eine Lösung, die keine Kopie benötigt:

```
void reversstring (char *s)
{
char *b, z;
     b = s+ strlen(s)-1;
     while (b>s)
     {
          z = *b;                    }  ≥   z=*b;
          *(b--) = *s;                      *(b)=*s;
          *(s++) = z;                       *(s)=z;
     }                                      s++;
}                                           b++;
```

Diese Lösung basiert darauf, daß ein String auch umgedreht wird, wenn das erste Zeichen mit dem letzten vertauscht wird, das zweite mit dem vorletzten usw.

Lösung zu Übung 5.

Lösung mit Indizes:

```
#include <iostream>
void mixstring (char *z, char *q1, char *q2)
{
int cz = 0, cq1 = 0, cq2 = 0;
     do
     {
          if (q1[cq1])   z[cz++] = q1[cq1++];
          if (q2[cq2])   z[cz++] = q2[cq2++];
     }    while ((q1[cq1]) || (q2[cq2]));
     z[cz] = 0;
}
int main ()
char quelle1[160], quelle2[160], ziel[320];
     cout << "Bitte String1 eingeben:";
     cin.getline (quelle1,160);
     cout << "Bitte String2 eingeben:";
     cin.getline(quelle2,160);
     mixstring(ziel,quelle1,quelle2);
     cout << "Vermischter String:" <<ziel << endl;
}
```

Lösung mit Dereferenzierungsoperator:

```
void mixstring (char *z, char *q1, char *q2)
{
     do
     {
          if (*q1)  *z++ = *q1++;
          if (*q2)  *z++ = *q2++;
          }    while ((*q1) | | (*q2));
     *z = 0;
}
```

Lösung zu Übung 6.

```
#include <iostream>

int len (char *s)
{
     int a = 0;
     while (s[a++]);
     return(a-1);
}

int ostrstr (char *a, char *b)
{
int x, y, z;

     for (x=0; x<(len(a)-len(b)); x++)
     {
          z = 1;
          for (y=0; y<len(b);y++)

          if (b[y]!=a[x+y])
          {
               z = 0;
               break;
          }
     if (z)
     return(x);
     }
return (-1);
}

int main ()
{
char quelle1[160],quelle2[160];
int index;

     cout << "Bitte String eingeben : ";
     cin.getline(quelle1,160);
     cout << "Bitte Suchstring eingeben: ";
     cin.getline(quelle2,160);

     index = ostrstr(quelle1,quelle2);
     if(index!=-1)

          cout<<"String an Pos "<<index<<" gefunden"<< endl;
     else
          cout << "Suchstring nicht enthalten!" << endl;
}
```

Lösung zu Übung 7.

```
void leftstr (char *z, char *q, int n)
{
for (n-1; n>=0; n--)
{
if(!*q) break;
*(z++) = *(q++);
}
*z = 0;
}
```

Der Bereichsschutz wurde durch die **if**- Anweisung realisiert, die bei Erreichen der Endekennung mit einem **break** die Schleife abbricht.

Lösung zu Übung 8.

```
void rightstr (char *z, char *q, int n)
{
for (n-1; n>=0; n--)
{
if ((int)len(q) -n>0)
q = q + len (q)-n;
leftstr (z, q, n);
}
```

rightstr benutzt die Funktionen **len** und **leftstr**, die schon in vorhergehenden Übungen programmiert wurden.

Lösung zu Übung 9.

midstr benutzt die Funktionen **len** und **leftstr**, die in Übung 6 und 7 programmiert wurden.

```
void midstr (char *z, char *q, int p, int n)
{
     if ((int)len(q)-p>0)
     q+ = p;
     leftstr (z, q, n);
}
```

Versuchen Sie zur Übung bei der oberen Funktion, anstelle von leftstr die Funktion rightstr zu benutzen.

Lösung zu Übung 10.

```
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;

void unter20(unsigned long wert, char *s, int eins)
{
     switch(wert)
     {
          case 1:
               if (eins)
                    strcat (s,"eins");
               else
                    strcat (s,"ein");
               break;
          case 2: strcat(s,"zwei"); break;
          case 3: strcat(s,"drei"); break;
          case 4: strcat(s,"vier"); break;
          case 5: strcat(s,"fuenf"); break;
```

```
            case 6: strcat(s,"sechs"); break;
            case 7: strcat(s,"sieben"); break;
            case 8: strcat(s,"acht"); break;
            case 9: strcat(s,"neun"); break;
            case 10: strcat(s,"zehn"); break;
            case 11: strcat(s,"elf"); break;
            case 12: strcat(s,"zwoelf"); break;
            case 13: strcat(s,"dreizehn"); break;
            case 14: strcat(s,"vierzehn"); break;
            case 15: strcat(s,"fuenfzehn"); break;
            case 16: strcat(s,"sechszehn"); break;
            case 17: strcat(s,"siebzehn"); break;
            case 18: strcat(s,"achtzehn"); break;
            case 19: strcat(s,"neunzehn"); break;
      }
}

void unter100(unsigned long wert, char *s, int eins)
{
      if(wert<20)
      {
            unter20(wert,s,eins);
            return;
      }

      if(wert%10)
      {
            unter20(wert%10,s,0);
            strcat(s,"und");
      }

      switch(wert/10)
      {
```

```
            case 2: strcat(s,"zwanzig"); break;
            case 3: strcat(s,"dreissig"); break;
            case 4: strcat(s,"vierzig"); break;
            case 5: strcat(s,"fuenfzig"); break;
            case 6: strcat(s,"sechzig"); break;
            case 7: strcat(s,"siebzig"); break;
            case 8: strcat(s,"achtzig"); break;
            case 9: strcat(s,"neunzig"); break;
      }
}


void unter1000(unsigned long wert, char *s, int eins)
{
      if (wert<100)
      {
            unter100(wert,s,eins);
            return;
      }

      unter20(wert/100,s,0);
      strcat(s,"hundert");

      if(wert%100)
      {
            unter100(wert%100,s,eins);
      }
      return;
}



void tausend(unsigned long wert, char *s)
```

```
{
     if(wert>=1000)
     {
          unter1000(wert/1000,s,0);
          strcat(s,"tausend");
     }

     if(wert%1000)
          unter1000(wert%1000,s,1);
}



void million(unsigned long wert, char *s)
{
     if(wert>=1000000)
     {
          int mil=wert/1000000;

          if(mil==1)
          {
               strcat(s,"einemillion");
          }
          else
          {
               unter1000(mil,s,0);
               strcat(s,"millionen");
          }

          wert%=1000000;
     }

     tausend(wert,s);
```

```
}


void milliarde(unsigned long wert, char *s)
{
     if(wert>=1000000000)
     {
          int mil=wert/1000000000;
          if(mil==1)
          {
               strcat(s,"einemilliarde");
          }
          else
          {
               unter1000(mil,s,0);
               strcat(s,"milliarden");
          }

          wert%=1000000000;
     }
     million(wert,s);
}


void toWord(unsigned long wert, char *s)
{
     if(wert==0)
     {
          strcat(s,"Null");
          return;
     }

     milliarde(wert,s);
```

```
     s[0]=toupper(s[0]);

     return;
}


int main ()
{
     unsigned long wert;
     char nummer[500];
     nummer[0]=0;
     cout << "Bitte Zahl eingeben:";
     cin >> wert;
     toWord(wert,nummer);
     cout << nummer << endl;

     return 0;
}
```