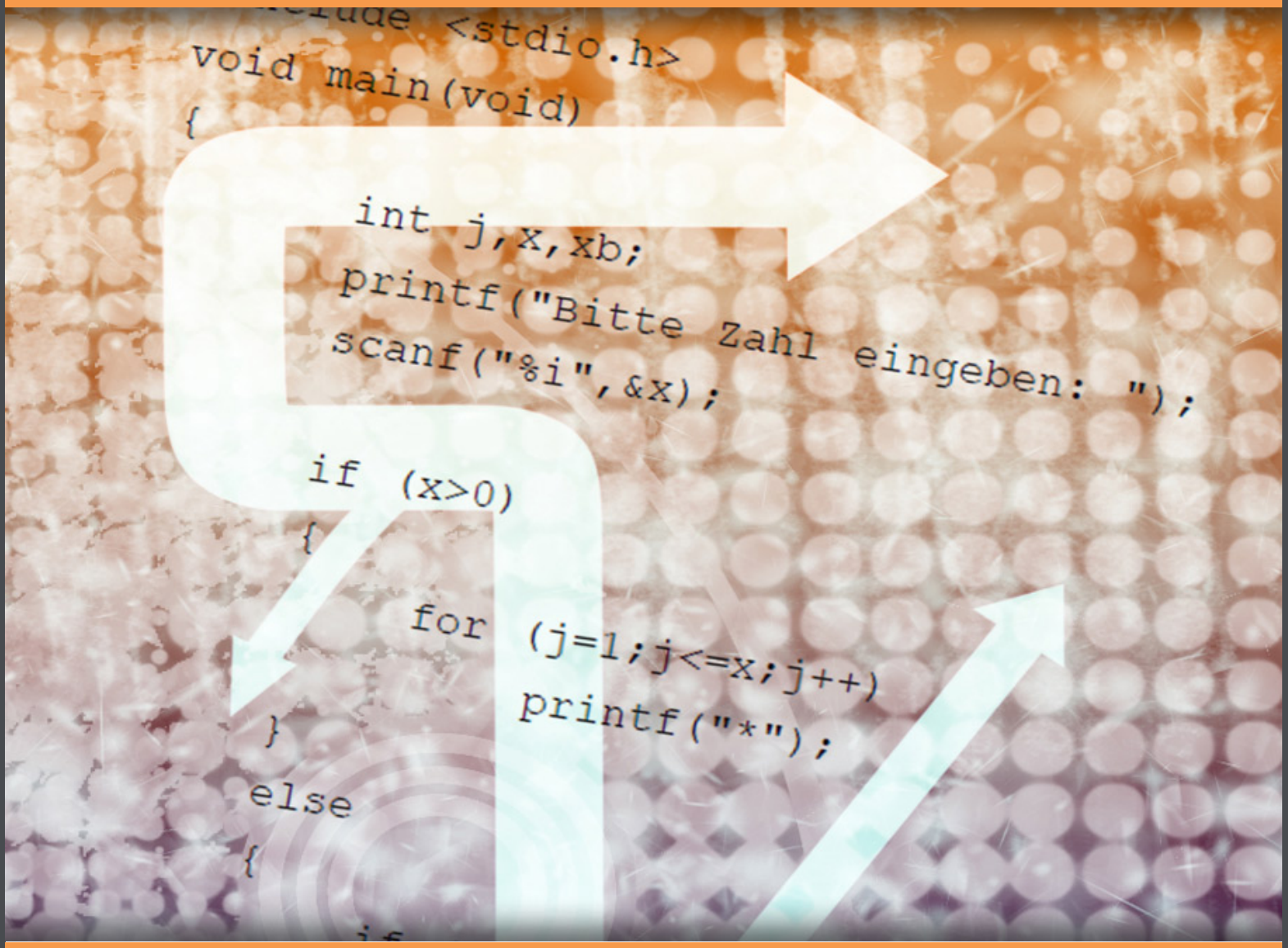
Kurs in der Programmiersprache C

Thomas Theis



Thomas Theis

Kurs in der Programmiersprache C

Kurs in der Programmiersprache C

ISBN 978-87-403-0064-2

Contents

1 Einführung

Die Programmiersprache C gibt es bereits seit 1973. Trotz Ihres "Alters" erfreut sie sich nach wie vor großer Beliebtheit und wird weiterhin intensiv genutzt. Außerdem stellt sie die Basis für viele weitere aktuelle Programmiersprachen dar, wie zum Beispiel C++, Java, PHP, C# (sprich: C-Sharp), Objective C und viele mehr. Es lohnt sich, sie zu erlernen. Sollten Sie später eine weitere Programmiersprache verwenden, so wird Ihnen dies aufgrund der Verwandtschaft unter den Sprachen wesentlich leichter fallen.

1.1 Software

Programmieren lernt man nur durch das Entwickeln und Testen von Programmen. Die Programme in der Sprache C müssen mithilfe eines C-Compilers in die Sprache des Rechners übersetzt werden. Falls Sie noch keinen C-Compiler haben: kein Problem, sie sind im Internet frei verfügbar. In Kapitel 19 werden Download, Installation und Benutzung von C-Compilern unter Windows und unter Ubuntu Linux erläutert.

1.2 Aufbau dieses Buchs

In jedem Kapitel werden Elemente der Programmiersprache C schrittweise vorgestellt und mit Unterstützung von Beispielen erklärt. Zum intensiven praktischen Lernen dienen die zahlreichen Übungsaufgaben, die selbständig zu lösen sind. Eine mögliche Lösung pro Aufgabe mit zahlreichen Kommentaren wird am Ende des Buchs vorgestellt.

1.3 Ein erstes Programm

Zunächst soll der Quellcode des Programms eingegeben werden:

```
/* Datei U0101.c */
#include <stdio.h>
void main(void)
{
       printf("Hallo\n);
       printf("Dies ist ein C Programm\n\n");
       printf("Mit C kann man rechnen\n");
       printf("13 * 7 = %i \n",13*7);
}
/* Ende */
```

Bei der Eingabe ist die Groß- bzw. Kleinschreibung zu beachten. Bestimmte Sonderzeichen wie die geschweiften Klammern { } und den umgekehrten Schrägstrich (Backslash) \ kann man eingeben, indem man die AltGr-Taste drückt, festhält und die jeweilige Taste drückt. Die Einrückung einer Zeile erreicht man zum Beispiel mithilfe von drei Leerzeichen.

Die Ausgabe sieht wie folgt aus:

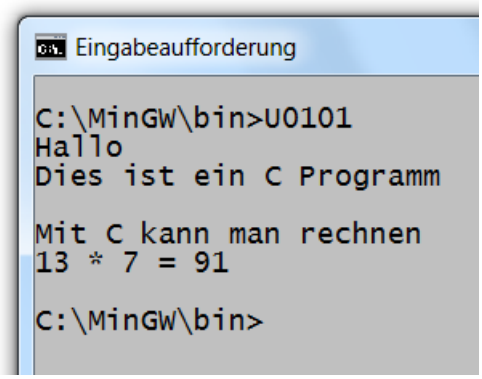


Abbildung 1.1: Ausgabe des Programms U0101

Die Erklärung der einzelnen Programm-Elemente:

- Kommentar: Bei den Angaben zwischen den Zeichen /* und */ handelt es sich um Kommentar. Hier können beliebige Texte zur Erläuterung des Programms eingegeben werden, die nur zur Kommentierung dienen und nicht übersetzt werden. Kommentare können über mehrere Zeilen gehen.
- #include <stdio.h>: Die wichtigsten Funktionen, wie z.B. printf(), stehen in der Standard-Bibliothek stdio.h. Diese sollte zu Beginn jedes Programms mit Hilfe des #include-Befehls eingebunden werden.
- void main(void): Ein C-Programm besteht aus einzelnen Funktionen. In jedem Programm gibt es mindestens eine Hauptfunktion, diese muss main genannt werden. Bei Aufruf des Programms beginnt der Ablauf bei main. Hinter jedem Funktionsnamen werden in Klammern Parameter eingegeben. Falls die Funktion keine Parameter hat, so wird dies durch void gekennzeichnet. Der Begriff void vor dem Funktionsnamen kennzeichnet die Tatsache, dass diese Funktion keinen Rückgabewert hat.
- Block: Die geschweiften Klammern { und } dienen zur Begrenzung eines Anweisungsblockes. Eine Funktion, wie hier main, muss in einen solchen Block geschrieben werden, damit Anfang und Ende der Funktion erkannt werden können.
- printf(): Die Funktion printf() dient zur Ausgabe von Daten auf dem Bildschirm. Hinter der Funktion folgen in Klammern die Parameter. Dies ist im einfachsten Fall der auszugebende Text, der in Anführungszeichen geschrieben werden muss.
- Escape-Sequenz: Eine Escape-Sequenz ist eine Zeichenfolge innerhalb eines auszugebenden Textes, die mit einem \ (Backslash) beginnt. Diese wird nicht ausgegeben, sondern dient zur Steuerung auf dem Bildschirm. Zum Beispiel bedeutet \n, dass hier eine neue Zeile begonnen wird.
- Berechnung eines Wertes: Mit Hilfe von printf können auch Werte berechnet werden. In der letzten Zeile hat printf zwei Parameter, als Erstes einen Text, als Zweites einen Ausdruck, der berechnet und ausgegeben wird. Zur Ausgabe wird außerdem noch ein Platzhalter innerhalb des Textes benötigt.
- Platzhalter: Der Platzhalter %i bewirkt, dass an dieser Stelle ein ganzzahliger Wert ausgegeben werden kann, der als zweiter Parameter folgt.
- Semikolon: In einem C - Programm muss hinter jeder Anweisung ein Semikolon stehen. Der Aufruf der Funktion printf() ist eine solche Anweisung. Eine Kommentarzeile oder die Vereinbarung der Hauptfunktion main() sind keine solchen Anweisungen.

Übung U0102:

Schreiben Sie ein Programm, das die nachfolgende Ausgabe auf dem Bildschirm erzeugt. Das Programm soll in der Datei U0102.c abgespeichert werden.

Dieser Text
besteht
aus
vier Zeilen.

Übung U0103:

Schreiben Sie ein Programm, das die nachfolgende Ausgabe auf dem Bildschirm erzeugt. Die Zahl 60 soll berechnet werden. Das Programm soll in der Datei U0103.c abgespeichert werden.

Das Ergebnis von zwölf mal fünf
ist 60

1.4 Die Funktion printf()

Innerhalb einer Zeichenkette, die mit Hilfe der Funktion printf() ausgegeben wird, können mehrere Platzhalter wie %i stehen. Sie müssen in Anzahl und Reihenfolge mit den berechneten Ausdrücken übereinstimmen. Diese Ausdrücke folgen als weitere Parameter, durch Komma getrennt. Ein Beispiel:

```
printf("5*5=%i, 6*8=%i, 7*3=%i \n",5*5,6*8,7*3);
```

Diese Programmzeile bewirkt die folgende Ausgabe:

```
5*5=25, 6*8=48, 7*3=21
```

Außer der Multiplikation können noch die Division (mit dem Zeichen /), die Addition (mit +) und die Subtraktion (mit -) durchgeführt werden. Falls mehrere Rechenzeichen (=Operatoren) vorkommen, so werden Multiplikation und Division vor Addition und Subtraktion durchgeführt.

Übung U0104:

Schreiben Sie ein Programm, das die nachfolgende Ausgabe auf dem Bildschirm erzeugt. Die Ergebnisse sollen berechnet werden. Das Programm soll in der Datei U0104.c abgespeichert werden. Ein Tip zur Erstellung des Programms: Schreiben Sie zuerst ein Programm, das die erste Ausgabezeile erzeugt und testen Sie es. Erst, wenn es erfolgreich läuft, sollte es erweitert werden.

```
Einige Rechnungen:
121 + 17 = 138
439 - 34 = 405
324 + 34 - 55 = 303
24 + 16 / 2 = 32
24 + 16 / 4 = 28
Teil 1: 37 - 5 * 3 = 22
Teil 2: 137 - 7 * 4 = 109
Teil 3: 28 + 49 / 7 = 35
48 / 3 * 2 = 32
48 / (3 * 2) = 8
```

2 Variablen und Datentypen

Zur Zwischenspeicherung von Werten verwendet man Variablen. Zu Beginn eines Programms müssen alle vorkommenden Variablen bekannt gemacht, d.h. deklariert werden.

2.1 Rechnen mit int-Variablen

Es soll folgendes Programm in die Datei U0201.c eingegeben, übersetzt und ausgeführt werden:

```
/* Datei U0201.c */
#include <stdio.h>
void main(void)
{
       int x,y;
       printf("\n\t\tR e c h e n p r o g r a m m  1\n\n");
       printf("\nBitte x eingeben: ");
       scanf("%i",&x);
       printf("\nBitte y eingeben: ");
       scanf("%i",&y);
       printf("\n\n %i + %i ist %i",x,y,x+y);
       printf("\n\n %i - %i ist %i",x,y,x-y);
       printf("\n\n %i * %i ist %i",x,y,x*y);
       printf("\n\n %i / %i ist %i",x,y,x/y);
}
```

Die Ausgabe sieht, in Abhängigkeit der Eingaben des Benutzers, wie folgt aus:

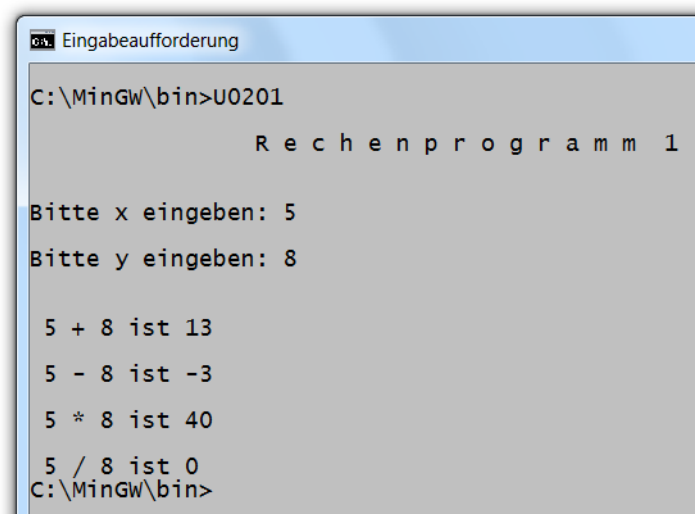


Abbildung 2.1: Ausgabe des Programms U0201

- int-Variablen: int-Variablen können ganze Zahlen aufnehmen und werden mit der Typbezeichnung int vereinbart. Danach folgen ein oder mehrere Variablen-Namen, durch Komma getrennt.
- Escape-Sequenz: \t bewirkt einen Vorschub um acht Stellen nach rechts
- Einlesen von Werten: Die Funktion scanf() dient zur Eingabe von Werten über die Tastatur. Sie arbeitet ähnlich wie printf. Vor dem Variablennamen muss allerdings das Zeichen & stehen.

Probieren Sie das Programm mit verschiedenen Werten für x und y aus. Beachten Sie das Ergebnis. int-Variablen haben einen Wertebereich von -2.147.483.648 bis 2.147.483.647. Falls dieser überschritten wird, ergeben sich keine sinnvollen Ausgaben mehr. Bei der Division von int-Zahlen werden die Nachkommastellen abgeschnitten, dazu später mehr.

2.2 Rechnen mit Fließkomma-Variablen

Das Programm aus der Datei U0201.c soll in die Datei U0202.c gespeichert und wie folgt verändert werden:

```
/* Datei U0202.c */
#include <stdio.h>
void main(void)
{
        float x,y;
        printf("\n\t\tR e c h e n p r o g r a m m  2\n\n");
        printf("\nBitte x eingeben: ");
        scanf("%f",&x);
```

```
        printf("\nBitte y eingeben: ");
        scanf("%f",&y);
        printf("\n\n %f + %f ist %f",x,y,x+y);
        printf("\n\n %f - %f ist %f",x,y,x-y);
        printf("\n\n %f * %f ist %f",x,y,x*y);
        printf("\n\n %f / %f ist %f",x,y,x/y);
}
```

- Fließkomma-Variablen: Sie können große Zahlen und Zahlen mit Nachkommastellen aufnehmen, von 3.4 * 10 hoch -38 bis 3.4 * 10 hoch +38. Die Typbezeichnung lautet float.
- Platzhalter: Der geeignete Platzhalter für float - Variablen ist %f. Es werden 6 Nachkommastellen angezeigt.

2.3 Formatierte Zahlenausgabe

Zur Verdeutlichung der formatierten Zahlenausgabe soll folgendes Programm in die Datei U0203.c eingegeben werden:

```
/* Datei U0203.c */
#include <stdio.h>
void main(void)
{
        float u_faktor, betrag;
        printf("\n\t\tW ä h r u n g s u m r e c h n u n g\n\n");
        printf("\nBitte Umrechnungsfaktor eingeben: ");
        scanf("%f",&u_faktor);
        printf("\nBitte Euro-Betrag eingeben: ");
        scanf("%f",&betrag);
        printf("\n%.2f Euro entspricht ", betrag);
        printf("\n%.2f in der Fremdwährung",betrag*u_faktor);
}
```

- Formatierung: Die Angabe .2 hinter dem Platzhalter bewirkt, dass die Zahl auf zwei Nachkommastellen gerundet wird.
- Variablennamen dürfen meist bis 31 Zeichen lang sein, aus den Zeichen A-Z, a-z, den Ziffern 0-9 und dem Zeichen _ bestehen. Sie dürfen nicht mit einer Ziffer beginnen.
- In C wird nach Groß- und Kleinschreibung unterschieden, dies gilt für Variablennamen, Befehle, Bibliotheksfunktionen.

Übung U0204:

Einen Temperaturwert, der in Celsius gegeben ist, kann man mit folgender Formel in Fahrenheit umrechnen: F = C * 9 / 5 + 32. Schreiben Sie ein Programm, das in der Lage ist, eine in Celsius eingegebene Temperatur umzurechnen und auszugeben (Datei U0204.c).

Übung U0205:

Schreiben Sie ein Programm, das den umgekehrten Vorgang ermöglicht (Datei U0205.c).

Übung U0206:

Schreiben Sie ein Programm, das die Anzahl der Sekunden eines Monates berechnet. Lassen Sie das Programm dazu die Anzahl der Sekunden pro Minute, die Anzahl der Minuten pro Stunde, die Anzahl der Stunden pro Tag und die Anzahl der Tage einlesen (Datei U0206.c).

Übung U0207:

Schreiben Sie ein Programm, das den Benzinverbrauch eines Autos in Liter pro 100 Km berechnet. Als Eingabe benötigt das Programm die verbrauchte Benzinmenge und die gefahrenen Km (Datei U0207.c).

2.4 Wertzuweisungen

Bisher wurde der Wert von berechneten Ausdrücken direkt auf dem Bildschirm ausgegeben. Mit Hilfe einer Zuweisung kann man einer Variablen einen solchen Wert geben. Dies kann eine einfache Zahl oder ein berechneter Ausdruck sein.

Dies soll mit folgendem Programm, das eingegeben werden soll, verdeutlicht werden:

```
/* Datei U0208.c */
#include <stdio.h>
void main(void)
{
        float summe, zahl;
        summe = 0.0;

        printf("\n1. Zahl eingeben: ");
        scanf("%f",&zahl);
        summe = summe + zahl;
        printf("\nZwischenergebnis = %.15f\n",summe);

        printf("\n2. Zahl eingeben: ");
        scanf("%f",&zahl);
        summe = summe + zahl;
        printf("\nZwischenergebnis = %.15f\n",summe);

        printf("\n3. Zahl eingeben: ");
        scanf("%f",&zahl);
        summe = summe + zahl;
        printf("\nEndergebnis = %.15f",summe);
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingaben des Benutzers, wie folgt aus:

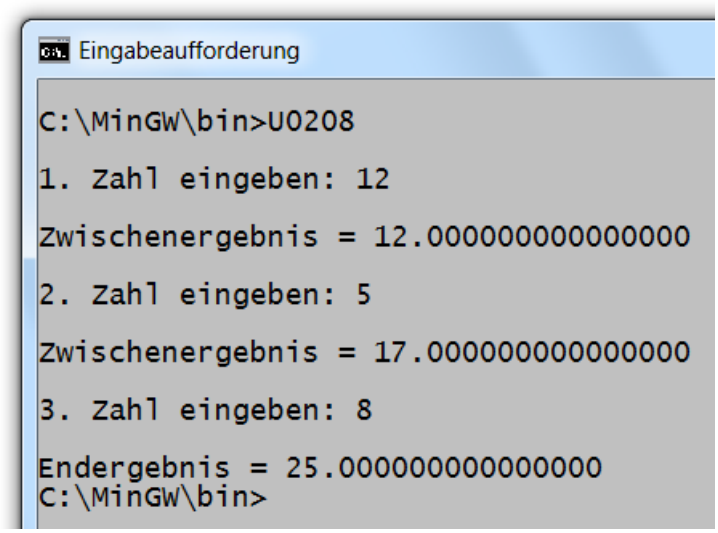


Abbildung 2.2: Ausgabe des Programms U0208

Mit der Anweisung summe = 0.0 wird der Variablen summe der Wert 0.0 zugewiesen. Mit der Anweisung summe = summe + zahl wird der Variablen summe ihr vorheriger Wert, erhöht um den Wert der Variablen zahl zugewiesen. Probieren Sie das Programm für verschiedene Zahlen aus.

Falls Sie Zahlen mit Nachkommastellen eingeben, können sich Ungenauigkeiten ergeben. Dies hat mit der Genauigkeit von Variablen zu tun. float-Variablen beanspruchen 4 Byte Speicherplatz im Rechner. Falls man genauer sein möchte, kann man auch den Typ double verwenden, dieser beansprucht 8 Byte Speicherplatz.

Verändern Sie das oben angegebene Programm, indem Sie statt des Typs float den Typ double verwenden (Datei U0209.c). Ein geeigneter Platzhalter für double-Variablen ist %lf (Abkürzung für long float) . Vergleichen Sie die Ergebnisse des Programms mit denen des vorherigen Programms.

2.5 Initialisierung

Der Variablen summe wurde als erstes ein bestimmter Wert (hier 0.0) zugewiesen. Dies nennt man Initialisierung einer Variablen. Ohne diese Initialisierung hätte das Programm u.U. keine sinnvollen Ergebnisse geliefert, da nicht jeder Compiler die deklarierten Variablen automatisch mit dem Wert 0 besetzt. Es hätte dann bei der Anweisung `summe = summe + zahl` kein sinnvoller vorheriger Wert zur Verfügung gestanden.

Verändern Sie Ihr Programm, indem Sie die Initialisierung löschen und probieren Sie es aus.

Übung U0210:

Verändern Sie das Programm aus Übung U0204, indem Sie double statt float-Variablen verwenden (Datei U0210.c).

Übung U0211:

Verändern Sie das Programm aus Übung U0206, indem Sie die Anzahl der Sekunden pro Minute, die Anzahl der Minuten pro Stunde, die Anzahl der Stunden pro Tag mit Hilfe von Wertzuweisungen im Programm angeben. Die Anzahl der Tage soll nach wie vor eingelesen werden (Datei U0211.c).

Übung U0212:

Schreiben Sie ein Programm, das die Anzahl der Tage jedes Quartals eines Jahres berechnet. Die Anzahl der Tage der verschiedenen Monate sollen mit Hilfe von Wertzuweisungen im Programm angegeben werden. Verwenden Sie geeignete Variablen zum Summieren der Tage eines Quartals (Datei U0212.c).

3 Schleifen mit for

Falls ein Teil eines Programms mehrfach ausgeführt werden soll, wird mit einer Schleife gearbeitet. Es gibt folgende Arten von Schleifen:

- for – Schleife
- do-while – Schleife
- while - Schleife

3.1 for - Schleife

Mit Hilfe des folgenden Programms werden die Zahlen 5 bis 10 auf dem Bildschirm ausgegeben:

```
/* Datei U0301.c */
#include <stdio.h>
void main(void)
{
        int x;
        for (x=5; x<11; x=x+1)
        {
        printf("Zahl: ");
        printf("%i\n",x);
        }
}
```

Die Ausgabe des Programms sieht wie folgt aus:

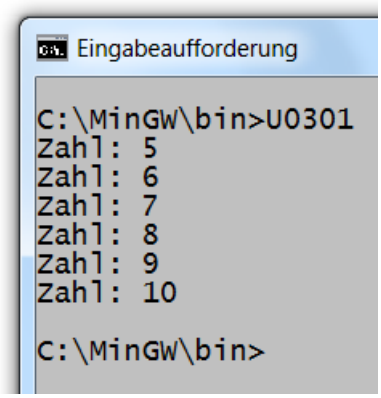


Abbildung 3.1: Ausgabe des Programms U0301

Eine for - Schleife besteht aus vier Komponenten:

- Der erste Ausdruck x=5 ist der Initialisierungsausdruck
- Der zweite Ausdruck x<11 legt das Laufkriterium der Schleife fest
- Der dritte Ausdruck x=x+1 wird bei jedem Schleifen-Durchlauf ausgeführt. Er führt normalerweise durch Inkrementierung oder Dekrementierung, d.h. Erhöhung oder Erniedrigung eines Wertes zur Verletzung des Laufkriteriums.
- Der eigentliche Schleifenrumpf, der mehrmals ausgeführt wird

Bei der for-Schleife ist die Anzahl der Schleifen - Durchläufe i.a. vor Schleifenbeginn bekannt. Falls die Schleife mehrere Anweisungen beinhaltet, wird Sie in einem Block geschrieben (Zeichen { und }). Falls die Schleife nur eine Anweisung beinhaltet, benötigen Sie keinen Block.

Übung U0302:

Erweitern Sie das o.a. Programm, indem Sie den Initialisierungsausdruck und das Laufkriterium vor der Schleife von Tastatur einlesen lassen (Datei U0302.c).

Übung U0303:

Erweitern Sie das Programm aus Übung U0302, indem Sie die Zahlen in der Schleife zusätzlich aufsummieren lassen. Das Ergebnis soll ausgegeben werden (Datei U0303.c).

Übung U0304:

Erweitern Sie das Programm aus Übung U0303, indem Sie zusätzlich den Mittelwert der Zahlen berechnen und ausgeben lassen (Datei U0304.c).

Übung U0305:

Schreiben Sie ein Programm, mit dessen Hilfe eine von Ihnen bestimmte Anzahl von Zahlen, die der Reihe nach von Tastatur eingelesen werden, aufsummiert werden kann (Datei U0305.c).

Übung U0306:

Schreiben Sie das Programm aus Datei U0301.c so um, dass Sie keinen Block in der Schleife benötigen (Datei U0306.c).

3.2 Vergleichsoperatoren

Der Ausdruck x < 11 ist ein Vergleich. Es gibt folgende Vergleichsoperatoren mit den angegebenen Bedeutungen:

<	kleiner als
>	größer als
<=	kleiner oder gleich
>=	größer oder gleich
!=	ungleich
==	gleich

Übung U0307:

Schreiben Sie das Programm aus Übung U0306 unter Verwendung des Operators <= (Datei U0307.c).

3.3 Mehrfache Schleifen

Man kann innerhalb einer Schleife wiederum eine Schleife durchlaufen. Dies wird anhand des folgenden Beispiels verdeutlicht:

```
/* Datei U0308.c */
#include <stdio.h>
void main(void)
{
        int x,y;
        for (x=3; x<6; x=x+1)
        {
             printf("\n");
             for (y=12; y<18; y=y+1)
             {
                  printf("%i ",x+y);
             }
        }
}
```

Die Ausgabe des Programms sieht wie folgt aus:

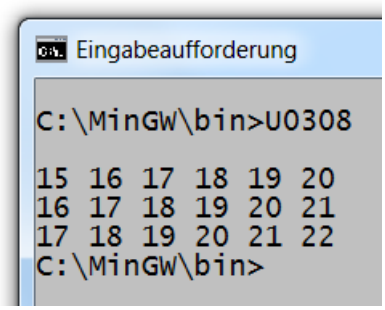


Abbildung 3.2: Ausgabe des Programms U0308

Die äußere Schleife wird für die x-Werte 3 bis 5 (=dreimal) durchlaufen. Die innere Schleife wird für die y-Werte 12 bis 17 (=sechsmal) durchlaufen. Die innerste printf-Anweisung wird also insgesamt 18 Mal durchlaufen.

Übung U0309:

Schreiben Sie ein Programm mit einer einfachen Schleife, das folgende Ausgabe auf dem Bildschirm erzeugt (Datei U0309.c): 24 27 30 33 36

Übung U0310:

Schreiben Sie ein Programm mit einer einfachen Schleife, das folgende Ausgabe auf dem Bildschirm erzeugt (Datei U0310.c):

35 34 33 32 31 30

Übung U0311:

Schreiben Sie ein Programm mit einer mehrfachen Schleife, das folgende Ausgabe auf dem Bildschirm erzeugt (Datei U0311.c):

24 27 30 33 36
25 28 31 34 37
26 29 32 35 38
27 30 33 36 39

Übung U0312:

Schreiben Sie ein Programm, mit dessen Hilfe die Fakultät einer eingegebenen Zahl berechnet werden kann. Die Fakultät von 6 berechnet sich aus 1*2*3*4*5*6 (Datei U0312.c):

4 Schleifen mit while und do-while

Falls man vor Beginn einer Schleife nicht weiß, wie oft sie durchlaufen werden soll, eignet sich die for - Schleife nicht mehr zur Wiederholung. Man sollte in diesem Fall entweder eine while - Schleife oder eine do - while - Schleife benutzen.

4.1 while - Schleife

Im nachfolgenden Programm wird der Einsatz einer while - Schleife verdeutlicht. Es werden Zahlen eingelesen und aufsummiert. Das Programm wird beendet, sobald die Summe der Zahlen größer oder gleich 50 ist.

```
/* Datei U0401.c */
#include <stdio.h>
void main(void)
{
        int zahl, summe=0;
        while (summe < 50)
        {
             printf("\nZahl eingeben: ");
             scanf("%i",&zahl);
             summe = summe + zahl;
             printf("Summe: %i",summe);
        }
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingaben des Benutzers, wie folgt aus:

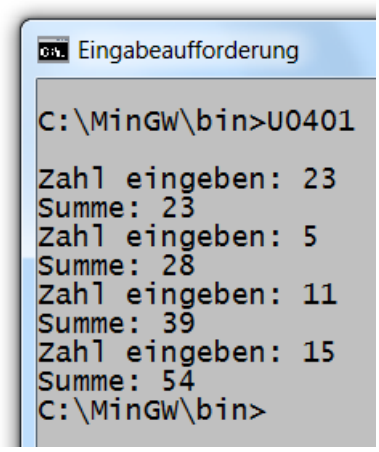


Abbildung 4.1: Ausgabe des Programms U0401

Eine while-Schleife besteht aus vier Komponenten:

- Initialisierung: "summe" wird mit 0 vorbesetzt. Dies entspricht einer Initialisierung der Schleife.
- Bedingung zur Wiederholung: Der Ausdruck "while (summe < 50)" bedeutet: Solange "summe" kleiner als 50 ist, soll die Schleife durchlaufen werden. Sollte hier eine Bedingung formuliert sein, die von Anfang an nicht zutrifft, so wird die Schleife nie durchlaufen. Die Anzahl der Wiederholungen ergibt sich erst während des Programmlaufes.
- Schleifenrumpf: Er wird wiederholt ausgeführt, solange die Bedingung zutrifft.
- Wertänderung der Variablen: Die eingelesenen Zahlen werden aufsummiert. Dies führt zu einer Wertänderung der Variablen "summe" und ggf. zum Ende der Wiederholung.

Bei allen Schleifen muss man darauf achten, dass es nicht zu endlosen Wiederholungen kommen kann. Dies ist der Fall, wenn die Bedingung zur Wiederholung immer zutrifft. Falls z.B. vergessen wird, die Variable "summe" in der Schleife zu verändern, würde dies zu einer Endlos-Schleife führen.

Übung U0402:

Schreiben Sie ein Programm, das für eine beliebige Anzahl von Zahlen das Quadrat berechnet und ausgibt. Das Programm soll beendet werden, sobald man eine Null eingegeben hat (Datei U0402.c).

Übung U0403:

Schreiben Sie ein Programm, mit dessen Hilfe eine eingegebene Zahl wiederholt halbiert und ausgegeben wird. Das Programm soll beendet werden, wenn das Ergebnis der Halbierung kleiner als 0.001 ist (Datei U0403.c).

Übung U0404:

Erweitern Sie das Programm aus Übung U0403, indem Sie die Zwischenergebnisse bei der Halbierung aufsummieren. Die Summe soll nach der Schleife ausgegeben werden (Datei U0404.c).

Übung U0405:

Erweitern Sie das Programm aus Übung U0404, indem Sie nicht den festen Wert 0.001 für die Bedingung verwenden, sondern einen Wert, den Sie über Tastatur einlesen lassen (Datei U0405.c).

4.2 do - while - Schleife

Die do - while - Schleife ähnelt der while - Schleife. Die Bedingung zur Wiederholung wird allerdings erst nach der Schleife gestellt, so dass die Schleife in jedem Fall mindestens einmal durchlaufen wird.

Im nachfolgenden Programm wird der Einsatz einer do - while - Schleife verdeutlicht. Es soll eine Zahl eingelesen werden, die in jedem Fall positiv sein soll.

```
/* Datei U0406.c */
#include <stdio.h>
void main(void)
{
        int zahl;
        do
        {
             printf("\nPositive Zahl eingeben: ");
             scanf("%i",&zahl);
        }
        while (zahl < 0);
        printf("Zahl: %i",zahl);
}
```

- Initialisierung: In einer do - while - Schleife gibt es keine eigentliche Initialisierung. Die Schleife wird in jedem Fall mindestens einmal durchlaufen.
- Bedingung zur Wiederholung: Der Ausdruck "while (zahl < 0)" bedeutet: Solange die Variable "zahl" kleiner als Null ist, soll die Schleife wiederholt werden.
- Schleifenrumpf: Er wird wiederholt ausgeführt, solange die Bedingung zutrifft.
- Wertänderung der Variablen: Einlesen der Variablen "zahl" in der Schleife. Dies führt zu einer Wertänderung der Variablen und ggf. zum Ende der Wiederholung.

Übung U0407:

Schreiben Sie das Programm aus Übung U0402 mit Hilfe einer do - while - Schleife (Datei U0407.c).

Übung U0408:

Schreiben Sie das Programm aus Übung U0403 mit Hilfe einer do - while - Schleife (Datei U0408.c).

Übung U0409:

Schreiben Sie ein Programm mit einer while - Schleife, das folgende Ausgabe auf dem Bildschirm erzeugt (Datei U0409.c):

24 27 30 33 36

Übung U0410:

Schreiben Sie ein Programm, mit dessen Hilfe die Fakultät einer eingegebenen Zahl berechnet werden kann. Die Fakultät von 6 berechnet sich aus 1*2*3*4*5*6 (Datei U0410.c). Achten Sie darauf, dass die eingelesene Zahl positiv ist. Verwenden Sie ausschließlich do - while - Schleifen.

5 Verzweigungen mit if-else

Es gibt in C zwei Möglichkeiten, innerhalb eines Programms eine Auswahl zu treffen. Zum einen mit Hilfe der if-Anweisung, zum anderen mit Hilfe der switch-Anweisung. Durch eine solche Auswahl-Anweisung wird entschieden, welcher Teil eines Programms durchlaufen werden soll.

5.1 if - Anweisung

Im folgenden Programm wird der Einsatz der if - Anweisung verdeutlicht. Es wird eine Zahl eingegeben. Falls die eingegebene Zahl positiv ist, wird die entsprechende Anzahl an Sternen auf dem Bildschirm ausgegeben.

```
/* Datei U0501.c */
#include <stdio.h>
void main(void)
{
        int j,x;
        printf("Bitte Zahl eingeben: ");
        scanf("%i",&x);
        if (x>0)
        {
             for (j=1;j<=x;j++)
                  printf("*");
        }
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingabe des Benutzers, wie folgt aus:

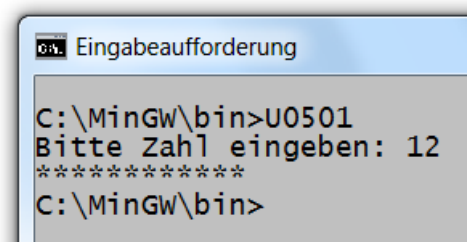


Abbildung 5.1: Ausgabe des Programms U0501

Hinter dem Befehl if steht in Klammern eine Bedingung. Falls diese Bedingung zutrifft, wird der anschließende Anweisungsblock in den geschweiften Klammern durchlaufen. Falls es sich nur um eine einzige Anweisung handelt, können wie bei den Anweisungen for, while, do-while, die geschweiften Klammern weggelassen werden. Hinweis: der Ausdruck j++ bedeutet das Gleiche wie j=j+1, d.h. der Wert von j wird um 1 erhöht.

Übung U0502:

Erweitern Sie das Programm aus Datei U0501.c. Es sollen nacheinander solange Zahlen eingegeben werden, bis eine Null eingegeben wurde (zur Erinnerung: do-while). Falls eine eingegebene Zahl positiv ist, wird die entsprechende Anzahl an Sternen auf dem Bildschirm ausgegeben (Datei U0502.c).

Übung U0503:

Schreiben Sie ein Programm, mit dessen Hilfe das Spiel "Zahlenraten" gespielt werden kann: Die erste Person gibt eine Zahl ein, danach werden 25 Zeilenvorschübe ausgeführt, damit die zu ratende Zahl nicht mehr sichtbar ist. Anschließend wird die zweite Person solange aufgefordert, Zahlen einzugeben, bis sie die Zahl erraten hat. Als Hilfestellung soll jedes Mal ausgegeben werden, ob die eingegebene Zahl größer oder kleiner als die zu ratende Zahl ist (Datei U0503.c).

Übung U0504:

Erweitern Sie das Programm aus Übung U0503, indem Sie eine Auswertung des Spielverlaufes machen (Datei U0504.c). Folgendes soll gezählt und ausgegeben werden:

- Die Anzahl der Versuche insgesamt
- Die Anzahl der Zahlen, die zu groß waren
- Die Anzahl der Zahlen, die zu klein waren

5.2 if-else- Anweisung

Falls man genau zwischen zwei Alternativen auswählen will, kann man die if-Anweisung durch ein else erweitern. Im folgenden Programm wird der Einsatz von else verdeutlicht.

Es wird eine Zahl eingegeben. Falls die eingegebene Zahl positiv ist, wird die entsprechende Anzahl an Sternen auf dem Bildschirm ausgegeben. Falls nicht, wird die entsprechende Anzahl an Doppelkreuzen auf dem Bildschirm ausgegeben.

```
/* Datei U0505.c */
#include <math.h>
#include <stdio.h>
void main(void)
{
       int j,x,xb;
       printf("Bitte Zahl eingeben: ");
       scanf("%i",&x);

       if (x>0)
       {
            for (j=1;j<=x;j++)
                 printf("*");
       }
       else
       {
            xb = abs(x);
            for (j=1;j<=xb;j++)
                 printf("#");
       }
}
```

Der Anweisungsblock hinter else wird ausgeführt, falls die erste Bedingung nicht zutrifft. Man hätte das gleiche auch mit zwei if-Anweisungen hintereinander erreichen können (if(x>0) und if(x<=0)). Allerdings wären dann jedes Mal zwei Bedingungen überprüft worden. Diese Lösung hätte das Programm langsamer gemacht.

Im Programm wurde die Funktion abs() benutzt, die den Betrag einer Zahl berechnet, also den Wert einer Zahl ohne Vorzeichen. Diese gehört zu den mathematischen Funktionen. Damit die Sammlung von mathematischen Funktionen innerhalb des Programms bekannt ist, muss die Datei math.h mit Hilfe des Befehls include eingebunden werden.

Übung U0506:

Schreiben Sie ein Programm, mit dessen Hilfe die folgende Buchstabenreihe ausgegeben wird: "AAABBBBB". Verwenden Sie dazu nur eine einzige Schleife. Innerhalb der Schleife soll je nach Wert ein "A" oder ein "B" ausgegeben werden (Datei U0506.c).

5.3 Geschachtelte if-Anweisung

Man kann innerhalb eines Anweisungsblockes nach einer if- oder if-else-Anweisung auch eine weitere if- oder if-else-Anweisung verwenden. Im folgenden Programm wird dies verdeutlicht:

```
/* Datei U0507.c */
#include <math.h>
#include <stdio.h>
void main(void)
{
        int j,x,xb;
        printf("Bitte Zahl eingeben: ");
        scanf("%i",&x);

        if (x>0)
        {
             for (j=1;j<=x;j++)
                  printf("*");
        }
        else
        {
             if (x<0)
             {
                  xb = abs(x);
                  for (j=1;j<=xb;j++)
                  printf("#");
             }
             else
                  printf("Es wurde 0 eingegeben");
        }
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingabe des Benutzers, wie folgt aus:

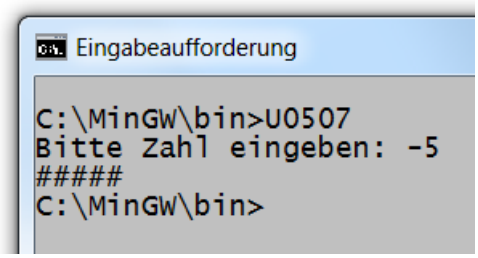


Abbildung 5.2: Ausgabe des Programms U0507

Falls die Zahl positiv ist, werden Sterne ausgegeben. Falls nicht, wird die Zahl weiter untersucht. Falls die Zahl negativ ist, werden Doppelkreuze ausgegeben. Falls die Zahl weder positiv noch negativ ist, wird ein Kommentar ausgegeben.

Übung U0508:

Schreiben Sie ein Programm, mit dessen Hilfe die folgende Buchstabenreihe ausgegeben wird: "AABBBBBCC". Verwenden Sie dazu nur eine einzige Schleife. Innerhalb der Schleife soll je nach Wert ein "A", "B" oder "C" ausgegeben werden (Datei U0508.c).

Übung U0509:

Schreiben Sie ein Programm, das zu einem eingegebenen Gehalt den Steuerbetrag berechnet und ausgibt (Datei U0509.c). In der folgenden Tabelle sind die Steuersätze angegeben.

Gehalt:	Steuersatz:
bis einschl. 12.000 Euro:	12%
von 12.000 bis einschl. 20.000 Euro:	15%
von 20.000 bis einschl. 30.000 Euro:	20%
über 30.000 Euro:	25%

Tabelle 5.1: Zu Übung U0509

Übung U0510:

Erweitern Sie das Programm aus Übung U0509, indem Sie eine Tabelle ausgeben. In der Tabelle sollen für alle Gehälter von 5.000 Euro bis 35.000 Euro in Schritten von 2.000 Euro folgende vier Werte ausgegeben werden: Gehalt, Steuersatz, Steuerbetrag, Gehalt abzüglich Steuerbetrag. Versehen Sie die Tabelle mit einer Überschrift und formatieren Sie die Tabelle, so dass die Spalten mit den Zahlen immer untereinander stehen (Datei U0510.c).

5.4 int - Division

In C gibt es zwei Operatoren bei der int - Division:

- Operator "/", Beispiel: 13 / 5 = 2, der Rest wird abgeschnitten
- Operator "%" (Modulo-Operator), Beispiel: 13 % 5 = 3, der Modulo - Operator liefert den Rest einer int-Division

Falls man wissen möchte, ob eine Zahl glatt durch eine andere Zahl teilbar ist, so kann man den Modulo-Operator verwenden. Das Ergebnis aus 15 % 5 ist 0, also ist 15 durch 5 teilbar.

Übung U0511:

Schreiben Sie ein Programm, das für eine Gruppe von positiven int-Zahlen untersucht, ob sie durch 2, 3, 5, 7 und 11 teilbar sind (Datei U0511.c). Lassen Sie die Unter- und die Obergrenze der Zahlengruppe von Tastatur einlesen. Die Eingabe der Zahlen soll solange wiederholt werden, bis sinnvolle Werte eingegeben wurden. Die Ausgabe soll in einer Tabelle mit Überschrift erfolgen, wie in folgender Abbildung:

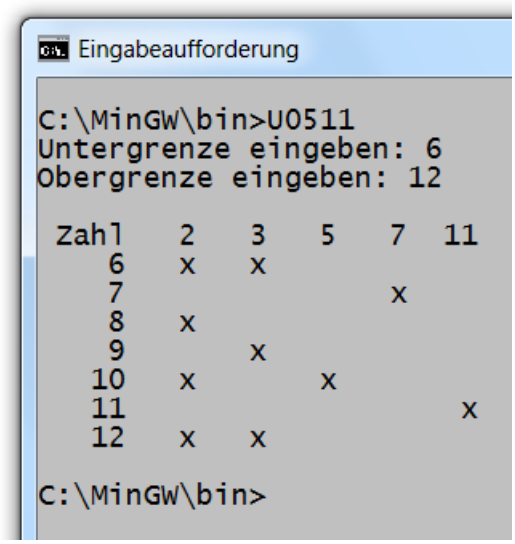


Abbildung 5.3: Ausgabe des Programms U0511

6 Mehr zu Verzweigungen

In diesem Kapitel werden erläutert:

- Logische Operatoren zur Verknüpfung von Bedingungen.
- Die switch-Anweisung zur mehrfachen Verzweigung.
- Der Datentyp char für einzelne Zeichen.

6.1 Logische Operatoren

Bisher bestanden Bedingungen sowohl bei der Wiederholung von Programmteilen (for, while, do-while) als auch bei der Auswahl (if, if-else) nur aus einfachen Ausdrücken. Es besteht auch die Möglichkeit, eine Aktion von mehreren Bedingungen abhängig zu machen. Dies geschieht mit Hilfe der logischen Operatoren UND (Zeichen: &&) bzw. ODER (Zeichen: ||), außerdem gibt es noch den Operator NICHT (Zeichen: !). Einige Beispiele dazu:

Mehrfach-Bedingung	x	(x<5)	y	(y<10)	Gesamt-Aussage
x<5 && y<10	4	wahr	9	wahr	wahr
x<5 && y<10	6	falsch	9	wahr	falsch
x<5 && y<10	4	wahr	11	falsch	falsch
x<5 && y<10	6	falsch	11	falsch	falsch
x<5 || y<10	4	wahr	9	wahr	wahr
x<5 || y<10	6	falsch	9	wahr	wahr
x<5 || y<10	4	wahr	11	falsch	wahr
x<5 || y<10	6	falsch	11	falsch	falsch
!(x<5)	4	wahr			falsch
!(x<5)	6	falsch			wahr

Tabelle 6.1: Logische Operatoren

Die Reihenfolge bei der Bearbeitung hängt von der Priorität der Operatoren ab. Eine Tabelle der häufigsten Operatoren, sortiert nach fallender Priorität:

Operator	Bedeutung
! -	logische Verneinung, negatives Vorzeichen
* /	Multiplikation, Division
+ -	Addition, Subtraktion
< > <= >=	kleiner, größer, kleiner gleich, größer gleich
== !=	gleich, ungleich
&&	logische UND-Verknüpfung
||	logische ODER-Verknüpfung

Tabelle 6.2: Priorität der Operatoren

Übung U0601:

Sind die folgenden Bedingungen wahr oder falsch ?

Nr.	Werte	Bedingung
1	int a=5, b=10;	while (a>0 && b!=10)
2	int a=5, b=10;	while (a>0 || b!=10)
3	int z=10, w=100;	if(z!=0 || z>w || w-z == 90)
4	int z=10, w=100;	if(z!=0 && z>w || w-z == 90)
5	double x=1.0, y=5.7;	if (x>=0.9 && y<=5.8)
6	double x=1.0, y=5.7;	if (x>=0.9 && !(y<=5.8))
7	int n1=1, n2=17;	while(n1>0 && n2>0 || n1>n2 && n2!=17)
8	int n1=1, n2=17;	while(n1>0 && (n2>0 || n1>n2) && n2!=17)

Übung U0602:

Schreiben Sie ein Programm, mit dessen Hilfe eine eingegebene Zahl wiederholt halbiert und ausgegeben wird. Das Programm soll beendet werden, wenn das Ergebnis der Halbierung kleiner als 0,001 ist. Es sollten allerdings maximal 10 Halbierungen durchgeführt werden, auch wenn das Ergebnis der Halbierung noch nicht kleiner als 0,001 ist. Verwenden Sie einen logischen Operator (Datei U0602.c).

Übung U0603:

Schreiben Sie das gleiche Programm unter der Bedingung, dass mindestens 15 Halbierungen durchgeführt werden, auch wenn das Ergebnis der Halbierung schon längst kleiner als 0,001 ist. Verwenden Sie einen logischen Operator (Datei U0603.c).

6.2 switch und break

Falls man eine Auswahl aus mehr als zwei Alternativen treffen möchte, kann man eine geschachtelte if-else Anweisung verwenden. Falls es sich um die Untersuchung eines ganzzahligen Ausdrucks handelt, gibt es auch noch die Möglichkeit, die switch-Anweisung einzusetzen. Im Folgenden ein Beispiel dazu:

```
/* Datei U0604.c */
#include <stdio.h>
void main(void)
{
        int x;
        printf("Bitte eine positive Ziffer eingeben\n");
        scanf("%i",&x);
        switch(x)
        {
             case 0:
             case 2:
             case 4:
             case 6:
             case 8:
                  printf("Gerade Ziffer\n");
                  break;
             case 1:
             case 3:
             case 5:
             case 7:
             case 9:
                  printf("Ungerade Ziffer\n");
                  break;
             default:
                  printf("Keine einzelne, positive Ziffer!\n");
        }
        printf("Programmende");
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingabe des Benutzers, wie folgt aus:

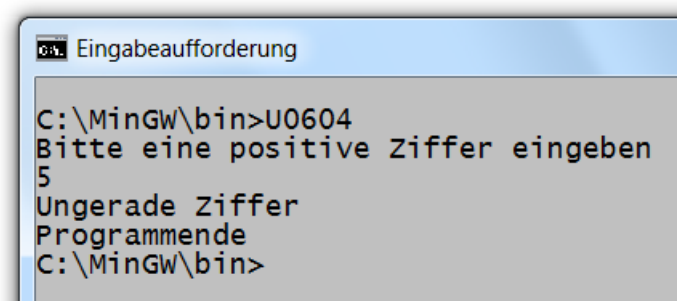


Abbildung 6.1: Ausgabe des Programms U0604

Hinter dem switch folgt in Klammern ein ganzzahliger Ausdruck. Dieser Ausdruck wird mit den Ausdrücken aus der case - Liste verglichen. Anschließend wird mit allen Befehlen, die hinter dem passenden Ausdruck gefunden werden, fortgefahren bis zum Ende des switch - Blockes. Falls kein passender Ausdruck gefunden wurde, wird hinter dem default fortgefahren, sofern vorhanden.

Somit würden bei Eingabe einer geraden Ziffer alle drei Texte ausgegeben. Dies wird allerdings durch die Anweisung break verhindert. Diese verursacht einen unmittelbaren Sprung hinter das Ende des switch - Blockes. Die break-Anweisung kann in for-, while- und do-while - Schleifen eingesetzt werden. Sie führt zu einem unmittelbaren Sprung hinter das Ende der jeweiligen Schleife.

Übung U0605:

Erstellen Sie ein Menü wie in der Menüleiste eines Anwendungsprogramms mit insgesamt sechs auswählbaren Menüpunkten. Dabei sollen folgende Schritte immer wieder durchlaufen werden (Datei U0605.c):

- Das Menü wird mit allen Menüpunkten ausgegeben, daneben die Zahlen 1 bis 6
- Der Benutzer kann eine Zahl eingeben
- Falls eine der Zahlen 1 bis 6 eingegeben wurde, wird der Name des entsprechenden Untermenüs ausgegeben
- Falls eine andere Zahl eingegeben wurde, wird eine Fehlermeldung ausgegeben
- Die Eingabe einer Null führt zum Ende des Programms

Übung U0606:

Schreiben Sie das Programm aus Übung U0602 mit Hilfe einer for-Schleife und der Anweisung break statt mit Hilfe eines logischen Operators (Datei U0606.c).

6.3 Zeichenkonstanten

Zur Speicherung eines einzelnen Zeichens dient der Datentyp char. Mit Hilfe des folgenden Programms wird:

- eine Variable vom Typ char deklariert
- der Variablen ein Zeichen zugewiesen
- dieses Zeichen auf dem Bildschirm ausgegeben
- der Wert dieses Zeichens im ASCII - Code ausgegeben
- der Variablen eine Zahl zugewiesen
- das zugehörige Zeichen auf dem Bildschirm ausgegeben
- die Zahl ausgegeben

```
/* Datei U0607.c */
#include <stdio.h>

void main(void)
{
        char a;

        a = 'Z';
        printf("%c\t",a);
        printf("%i\n",a);

        a = 82;
        printf("%c\t",a);
        printf("%i\n",a);
}
```

Die Ausgabe des Programms sieht wie folgt aus:

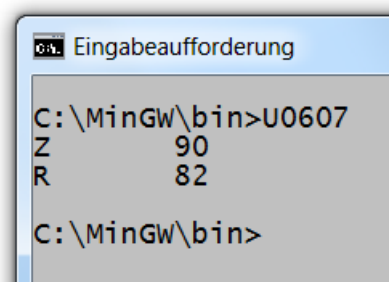


Abbildung 6.2: Ausgabe des Programms U0607

Zeichenkonstanten werden in einfache Hochkommata eingeschlossen. Der geeignete Platzhalter für Zeichenkonstanten ist %c. Eine Besonderheit bei Zeichenkonstanten ist die Tatsache, dass sie als ganze Zahlen gespeichert werden. Die Zahl entspricht der Position des Zeichens im ASCII-Code. Dies führt dazu, dass sowohl ganze Zahlen als auch Zeichenkonstanten als Zeichen oder als Wert ausgegeben werden können

Der ASCII - Code umfaßt 256 Zeichen (von 0 bis 255), dabei sind die ersten 128 (von 0 bis 127) einheitlich, die zweiten 128 (von 128 bis 255) abhängig von der benutzten Umgebung.

Übung U0608:

Schreiben Sie ein Programm, das die Zeichen mit den Werten 65 bis 90 und 97 bis 122 ausgibt (Datei U0608.c).

Übung U0609:

Schreiben Sie ein Programm, das eine übersichtliche Tabelle der ASCII-Zeichen von 0 bis 255 ausgibt (Datei U0609.c). Es sollte jeweils die Nummer des Zeichens und das Zeichen selber ausgegeben werden. Innerhalb einer Zeile sollten jeweils 8 Zeichen und ihre Codes stehen. Die Zeichen 7 bis 10, 13, 26 und 27 sollten dabei ausgelassen werden, da sie u.a. zur Bildschirmsteuerung dienen.

7 Funktionen

In den bisherigen Programmen wurden bereits Bibliotheks - Funktionen für bestimmte Aufgaben verwendet. Darüber hinaus kann man eigene Funktionen schreiben und verwenden. Dies ist aus folgenden Gründen sinnvoll:

- Funktionen erfüllen Aufgaben, die man in den eigenen Programme häufig verwenden, aber nur einmal entwerfen möchte
- Funktionen dienen der Modularisierung eines Programms. Unter Modularisierung versteht man die Aufteilung von größeren Programmen in kleinere, übersichtliche Bestandteile, die unabhängig voneinander entworfen und später zusammengefügt werden können.

7.1 Einfache Funktion

Im Folgenden wird der Einsatz einer einfachen Funktion an einem Beispiel verdeutlicht.

```
/* Datei U0701.c */
#include <stdio.h>

void stern(void)
{
        int x;
```

```
        for (x=1;x<26;x++)
             printf("*");
        printf("\n");
}

void main(void)
{
        int x,y;
        stern();
        printf("x eingeben: ");
        scanf("%i",&x);
        stern();
        printf("y eingeben: ");
        scanf("%i",&y);
        stern();
        printf("Summe: %i\n",x+y);
        stern();
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingabe des Benutzers, wie folgt aus:

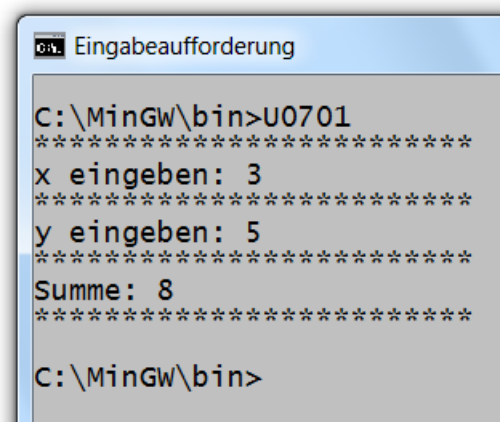


Abbildung 7.1: Ausgabe des Programms U0701

Im Programm wird die Funktion stern() vor der Funktion main() definiert und in der Funktion main() mehrmals aufgerufen. Die vollständige Definition einer Funktion beinhaltet:

- Rückgabewert der Funktion: In diesem Fall hat die Funktion stern() keinen Rückgabewert, daher die Angabe void vor dem Namen
- Name der Funktion: Wie bei Variablen dürfen sie aus maximal 31 Zeichen bestehen. Es dürfen A-Z, a-z, 0-9 und Unterstrich _ beinhaltet sein. Sie sollten nicht mit einem Unterstrich beginnen, da viele Bibliotheksfunktionen mit einem Unterstrich beginnen

- Parameterliste: Diese steht in Klammern hinter dem Funktionsnamen. Der Funktion stern() werden keine Parameter übergeben, daher die Angabe void in Klammern.
- Anweisungsblock: Dieser wird, wie jeder Block, in geschweiften Klammern geschrieben. Innerhalb der Funktion können eigene Variablen deklariert werden. Diese Variablen sind nur innerhalb der Funktion gültig (lokal). Beispiel: Es wurde zweimal der Name x verwendet, diese beiden Variablen sind unabhängig voneinander.

Übung U0702:

Schreiben Sie ein Programm, in dem die Funktion box() definiert und dreimal aufgerufen wird. Die Funktion zeichnet eine rechteckige Box auf den Bildschirm. Sie können dazu die ASCII - Zeichen 186, 187, 188, 200, 201, 205 und das Leerzeichen verwenden (Datei U0702.c).

7.2 Parameter

Die Funktion stern() und die Funktion box() haben genau festgelegte Aufgaben erfüllt. Falls diese Funktionen flexibler arbeiten sollen, so sollten sie mit Parametern aufgerufen werden. Dies soll am Beispiel der Funktion stern() gezeigt werden:

```
/* Datei U0703.c */
#include <stdio.h>

void stern(int anzahl)
{
       int x;
       for (x=1;x<anzahl;x++)
            printf("*");
       printf("\n");
}

voi]d main(void)
{
       int x,y,z;
       stern(25);
       printf("x eingeben: ");
       scanf("%i",&x);
       stern(30);
       printf("y eingeben: ");
       scanf("%i",&y);
       z=35;
       stern(z);
       printf("Summe: %i\n",x+y);
       z+=5;
       stern(z);
}
```

- Definition der Funktion: In Klammern hinter dem Funktionsnamen steht nun eine int-Variable als Parameter. Falls man mehrere Parameter verwendet, so müssen sie durch Kommas voneinander getrennt werden. Jede Variable muss einzeln mit Datentyp angegeben werden.
- Aufruf der Funktion: Innerhalb von main() wird die Funktion stern() mehrmals mit verschiedenen Parametern aufgerufen. Der Datentyp muss mit dem der Variablen in der Parameter-Liste übereinstimmen. Bei Verwendung von mehreren Parametern muss auch die Reihenfolge übereinstimmen.
- Parameterübergabe: Bei Aufruf der Funktion wird der Variablen in der Parameter-Liste der Wert des Parameters im Aufruf zugewiesen. Beim ersten Aufruf hat anzahl den Wert 25, beim zweiten den Wert 30 usw.

Übung U0704:

Schreiben Sie ein Programm, in dem in der Funktion main() zwei double Zahlen eingelesen werden. Außerdem soll eine Funktion mult() definiert und aufgerufen werden, die diese beiden Zahlen multipliziert und das Ergebnis mit Kommentar auf dem Bildschirm ausgibt (Datei U0704.c).

Übung U0705:

Schreiben Sie ein Programm, in dem drei float - Zahlen eingelesen werden. Diese Zahlen sollen einer Funktion mw() übergeben werden, die den Mittelwert der Zahlen berechnet und ausgibt (Datei U0705.c).

7.3 Rückgabewert

Funktionen können auch einen Rückgabewert haben, d.h. Ergebnisse an die aufrufende Funktion zurückliefern. Im folgenden Programm werden einer Funktion max() zwei int-Zahlen übergeben. Die Funktion liefert als Rückgabewert die größere der beiden Zahlen zurück an die aufrufende Funktion.

```
/* Datei U0706.c */
#include <stdio.h>
int max(int x, int y)
{
       if (x>y)
            return(x);
       else
            return (y);
}

void main(void)
{
       int a=6, b=5, c;
       c = max(a,b);
       printf("Das Maximum von %i und %i ist %i", a,b,c);
}
```

Die Ausgabe des Programms sieht wie folgt aus:

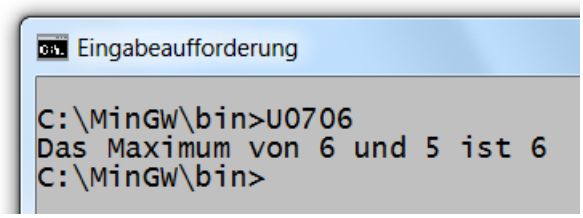


Abbildung 7.1: Ausgabe des Programms U0706

Der Datentyp der Funktion muss mit dem Datentyp der Variablen übereinstimmen, die den Rückgabewert zugewiesen bekommt (c und max sind beide vom Typ int). Die Anweisung return verursacht den unmittelbaren Rücksprung in die aufrufende Funktion, der Ausdruck in Klammern wird als Wert zurückgeliefert.

Übung U0707:

Verändern Sie das Programm aus Übung U0705. Die Ausgabe der Ergebnisse des Funktionsaufrufes soll im Hauptprogramm stattfinden. Die Funktion soll also das Ergebnis als Rückgabewert an das Hauptprogramm zurückliefern (Datei U0707.c).

7.4 Prototypen

Bisher wurden die Funktionen vor dem Programmteil definiert, von dem sie aufgerufen wurden. Falls man die Reihenfolge verändern möchte, muss man mit Prototypen arbeiten. Im folgenden Programm wird dieses verdeutlicht.

```
/* Datei U0708.c */
#include <stdio.h>
int max(int, int);

void main(void)
{
       int a=6, b=5, c;
       c = max(a,b);
       printf("Das Maximum von %i und %i ist %i", a,b,c);
}

int max(int x, int y)
{
       if (x>y) return(x);
       else return (y);
}
```

Vor main() steht der Prototyp der Funktion max(), d.h. Datentyp, Name, Datentypen der Parameter in Klammern und ein Semikolon am Ende. Die Namen der Parameter müssen noch nicht bekannt sein. Hinter main() steht die Definition der Funktion max(). Ohne Prototyp würde der Aufruf von max() im Hauptprogramm zu einem Fehler führen, da die Funktion noch unbekannt ist.

Übung U0709:

Verändern Sie das Programm aus Übung U0707. Schreiben Sie die Funktion unterhalb des Hauptprogramms (Datei U0709.c).

Übung U0710:

Schreiben Sie eine Funktion vokal(). Dieser Funktion wird ein Zeichen als Parameter übergeben. Falls dieses Zeichen ein Vokal ist, liefert Sie eine 1 zurück, sonst eine 0. Innerhalb der Funktion soll die switch-Anweisung verwendet werden. Testen Sie Ihre Funktion mit einem geeigneten Hauptprogramm (Datei U0710.c).

Übung U0711:

Schreiben Sie eine Funktion vorzeichen(). Dieser Funktion wird eine double - Zahl übergeben. Falls die Zahl positiv ist, so liefert sie +1 zurück, falls die Zahl negativ ist, so liefert sie -1 zurück, falls die Zahl gleich Null ist, liefert sie 0 zurück. Testen Sie Ihre Funktion mit einem geeigneten Hauptprogramm (Datei U0711.c).

Übung U0712:

Erweitern Sie die Funktion box() aus Übung U0702. Höhe und Breite der Box sollen als Parameter übergeben werden, so dass eine beliebig große Box auf dem Bildschirm ausgegeben werden kann. Wählen Sie sinnvolle Grenzwerte, die nicht über- bzw. unterschritten werden sollen.

Höhe und Breite der Box sollen im Hauptprogramm beliebig oft von Tastatur eingelesen werden können. Nach dem Zeichnen jeder Box soll der Benutzer gefragt werden, ob die nächste Box gezeichnet werden soll oder das Programm beendet werden soll (Eingabe = 1 bedeutet "Weitermachen", Eingabe = 2 bedeutet "Beenden"). Als Antwort darf der Benutzer nur 1 oder 2 eingeben (Datei U0712.c).

8 Eindimensionale Felder

Falls man eine ganze Reihe von Variablen des gleichen Typs speichern möchte, kann man entweder für jede Variable einen eigenen Namen vergeben oder alle Variablen in einem Feld speichern. Dieses Feld besteht aus einem Namen und einer Größenangabe.

8.1 Deklarieren, Werte zuweisen

Mit Hilfe der Deklaration: "double temperatur[5]" wird Speicherplatz für insgesamt 5 double-Werte reserviert. Die einzelnen Elemente des Feldes haben die Namen temperatur[0] bis temperatur[4].

Mit Hilfe des folgenden Programms werden Werte für die Elemente eines Feldes eingelesen und ausgegeben:

```
/* Datei U0801.c */
#include <stdio.h>
void main(void)
{
        double temperatur[5];
        int i;

        for (i=0;i<5;i++)
        {
```

```
            printf("Wert %i eingeben: ",i);
            scanf("%lf",&temperatur[i]);
        }

        for (i=0;i<5;i++)
            printf("\nWert %i: %lf",i,temperatur[i]);
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingabe des Benutzers, wie folgt aus:

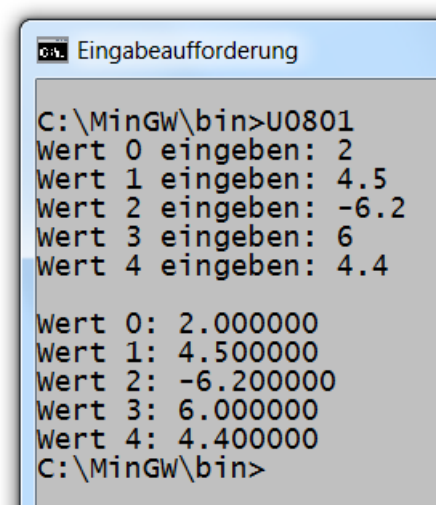


Abbildung 8.1: Ausgabe des Programms U0801

Übung U0802:

Schreiben Sie ein Programm (Datei U0802.c), in dem:

- ein Feld für 5 double - Zahlen deklariert wird
- alle Werte für dieses Feld von Tastatur eingelesen werden
- die Werte aufsummiert werden und die Summe ausgegeben wird
- der Mittelwert berechnet und ausgegeben wird

Übung U0803:

Es soll die kleinste Zahl des Feldes aus Übung U0802 gesucht und ausgegeben werden (Datei U0803.c). Zum Ablauf:

- Als erstes wird angenommen, dass das erste Element des Feldes die kleinste Zahl ist
- Anschließend werden die weiteren Feldelemente mit dem Minimum verglichen. Falls eines der Elemente kleiner als das Minimum ist, wird dieses Element das neue Minimum

Übung U0804:

Kopieren Sie die Elemente des Feldes aus Übung U0802 in ein zweites Feld. Die Reihenfolge der Elemente soll dabei umgedreht werden (Datei U0804.c). Lassen Sie beide Felder nebeneinander ausgeben.

Übung U0805:

Es soll die Position des ersten Vorkommens der kleinsten Zahl des Feldes aus Übung U0802 gesucht und ausgegeben werden (Datei U0805.c).

- Beispielfeld: 0.3, -5, 8.63, -7.66, -7.66
- Kleinste Zahl: -7.66
- Position des ersten Vorkommens der kleinsten Zahl innerhalb des Feldes: 3

Zum Ablauf:

- Als erstes wird angenommen, dass die Position des kleinsten Elementes 0 ist
- Anschließend werden die weiteren Feldelemente mit dem Minimum verglichen. Falls eines der Elemente kleiner als das Minimum ist, wird dessen Position die Position des kleinsten Elementes

Übung U0806:

Es soll die Entwicklung des Mittelwertes betrachtet werden. Dazu werden in einem zusätzlichen Feld die Mittelwerte in folgender Form abgespeichert: Im ersten Element steht die erste Zahl, im zweiten Element der Mittelwert der ersten beiden Zahlen, im dritten Element der Mittelwert der ersten drei Zahlen usw. (Datei U0806.c). Lassen Sie beide Felder nebeneinander ausgeben.

Übung U0807:

Lösen Sie Übung U0803 mit Hilfe einer Funktion vglmin(). Dieser Funktion übergeben Sie zwei Zahlen, die miteinander verglichen werden. Sie liefert die kleinere der beiden Zahlen als Ergebnis zurück (Datei U0807.c).

Übung U0808:

Es soll ein int-Feld der Größe 5 untersucht werden. Die Position des größten Elementes des Feldes soll gesucht und ausgedruckt werden. Falls es mehrere größte Elemente des Feldes gibt, sollen alle Positionen gesucht und ausgedruckt werden (Datei U0808.c). Sie benötigen dazu ein zusätzliches Feld, in dem die Positionen gespeichert werden können.

9 Zeichenketten

In vielen Programmen kommen Zeichenketten (Strings) zur Speicherung von Texten vor. Die Programmiersprache C bietet keinen eigenen Datentyp für Zeichenketten, sondern verwendet Felder von Zeichenkonstanten. Darüber hinaus gibt es in den Bibliotheken viele Funktionen, die der Verarbeitung von Zeichenketten dienen.

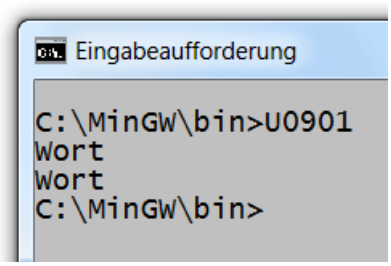
9.1 Aufbau

Das folgende Programm verdeutlicht den Aufbau einer Zeichenkette aus einzelnen Zeichenkonstanten:

```
/* Datei U0901.c */
#include <stdio.h>

void main(void)
{
        char tx[10];
        int i;

        tx[0] = 'W';
        tx[1] = 'o';
        tx[2] = 'r';
```

```
        tx[3] = 't';
        tx[4] = '\0';

        for (i=0;i<10;i++)
             printf("%c",tx[i]);
        printf("\n%s",tx);
}
```

Die Ausgabe des Programms sieht wie folgt aus:



bbildung 9.1: Ausgabe des Programms U0901

Die restlichen Elemente des Feldes tx wurden nicht belegt. Falls man sie sich einzeln anzeigen läßt, werden, wie in Zahlenfeldern, zufällige Inhalte ausgegeben. Dies können sichtbare oder nicht sichtbare Zeichen sein. Den Platzhalter %s kann man zur Ausgabe des ganzen Feldes von Zeichen nutzen. Das Zeichen ‚\0' (binäre Null) begrenzt eine Zeichenkette. Alle Zeichenkettenfunktionen beachten automatisch diese Besonderheit. Falls man einen Text mit 20 Zeichen speichern möchte, so muss man mindestens ein Feld der Größe 21 verwenden, damit das Zeichen ‚\0' angehängt werden kann.

9.2 Pufferung der Eingabe

Das Standard-Eingabegerät bei C-Programmen ist die Tastatur. Diese Standard-Eingabe hat einen Puffer, d.h. die Eingaben werden zunächst zwischengespeichert und erst anschliessend den Variablen zugewiesen. Falls der Benutzer genau die richtige Menge an Eingaben macht, so stellt dies kein Problem dar. Sollte er allerdings zu viele Eingaben machen, so stehen die überzähligen Eingaben noch im Puffer und werden späteren Eingaben automatisch zugewiesen. Dies kann ein erwünschter Effekt sein, meist ist dies aber nicht der Fall. Der Aufruf der Funktion fflush(stdin) führt zum Löschen des Puffers der Standard-Eingabe. Somit kann der unerwünschte Effekt der Pufferung vermieden werden.

Ein Beispiel: Falls im nachfolgenden Programm bei der ersten Eingabe zwei ganze Zahlen eingegeben werden (durch Leerzeichen getrennt), so wird die zweite Zahl unmittelbar für die zweite Eingabe verwendet, auch wenn dies nicht geplant ist. Falls nach der ersten Eingabe allerdings der Puffer geleert wird (mit `fflush(stdin)`), so wird der Benutzer auch bei fehlerhafter erster Eingabe korrekt zur zweiten Eingabe aufgefordert. Die überzähligen Zeichen oder Zahlen der ersten Eingabe sind vorher gelöscht worden.

```
/* Datei U0902.c */
#include <stdio.h>

void main(void)
{
       int a;

       printf("ganze Zahl eingeben: ");
       scanf("%i",&a);
       /* fflush(stdin); */
       printf("%i\n",a);

       printf("ganze Zahl eingeben: ");
       scanf("%i",&a);
       printf("%i\n",a);

       printf("\n");
}
```

9.3 Eingabe von Zeichen und Zeichenketten

Die Funktion getchar() dient zum Einlesen von einzelnen Zeichen. Dabei kann es sehr schnell zur Eingabe von überzähligen Zeichen kommen, daher sollte hier auch der Aufruf von fflush(stdin) zum Löschen des Puffers erfolgen. Die Funktion gets() dient zum Einlesen von ganzen Zeichenketten, bestehend aus mehreren Worten. Im nachfolgenden Programm werden einige Zahlen, Zeichen bzw. Zeichenketten eingegeben und anschließend zur Kontrolle ausgegeben.

```
/* Datei U0903.c */
#include <stdio.h>

void main(void)
{
       char tx[10];
       char z;
       int a, i;
       double x;

       for(i=1; i<=3; i++)
       {
            printf("Zeichen eingeben: ");
            z = getchar();
            fflush(stdin);
            printf("%c\n",z);
       }
```

```
        printf("Zahl eingeben: ");
        scanf("%lf",&x);
        fflush(stdin);
        printf("%lf\n",x);

        for(i=1; i<=3; i++)
        {
             printf("Zeichenkette eingeben: ");
             gets(tx);
             printf("%s\n",tx);
        }

        printf("ganze Zahl eingeben: ");
        scanf("%i",&a);
        printf("%i\n",a);

        printf("\n");
}
```

Die Funktion getchar() weist das eingelesene Zeichen der Variablen (hier z) zu. Die Funktion gets() arbeitet etwas anders: Ihr wird das Feld von Zeichen, in dem die eingegebene Zeichenkette gespeichert werden soll (hier tx), als Parameter übergeben.

Unter MS Visual C tritt bei der gemischten Eingabe von Zahlen und Zeichenketten ein merkwürdiger Effekt auf: Bei der Eingabe der double-Zahl kann es zur Eingabe von überzähligen Zeichen kommen, die die nachfolgende Eingabe der Zeichenkette beeinflussen. Damit dies nicht auftritt, wird in diesem Programm auch nach der Eingabe der double-Zahl der Puffer geleert. Damit ist man in jedem Fall auf der "sicheren Seite".

9.4 Weitere Zeichenkettenfunktionen

Aus den zahlreichen Funktionen zur Verarbeitung von Zeichenketten (Strings) werden im folgenden Programm einige wichtige vorgestellt. Zur Benutzung muss die Header-Datei string.h eingebunden werden. Es handelt sich um die Funktionen:

- `strlen()`: zur Messung der Länge einer Zeichenkette
- `strcpy()`: zur Zuweisung einer Zeichenkette
- `strcat()`: zum Anhängen einer Zeichenkette an eine vorhandene Zeichenkette
- `strcmp()`: zum Vergleich zweier Zeichenketten

```
/* Datei U0904.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char txa[10], txb[10];
        int laenge,vgl;

        printf("Text A eingeben: ");
        gets(txa);
        printf("\nText A: %s",txa);
        laenge = strlen(txa);
        printf("\n%i",laenge);

        strcpy(txb,"Soft");
        printf("\nText B: %s",txb);
        printf("\n%i",strlen(txb));

        strcat(txb,"ware");
        printf("\nText B neu: %s",txb);
        printf("\n%i",strlen(txb));
```

```
        vgl = strcmp(txa,txb);
        printf("\n%i\n",vgl);
        if (vgl < 0)
             printf("Text A im ASCII-Code vor Text B");
        else
        {
             if (vgl > 0)
                  printf("Text A im ASCII-Code hinter Text B");
             else
                  printf("Text A = Text B");
        }

        printf("\n");
}
```

Die Ausgabe des Programms sieht, in Abhängigkeit der Eingabe des Benutzers, wie folgt aus:

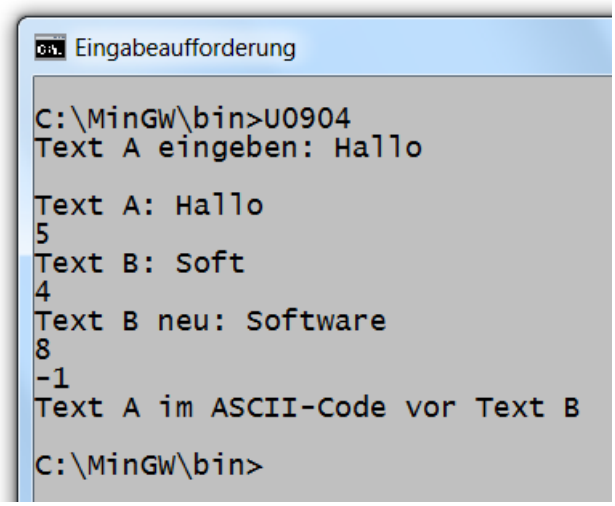


Abbildung 9.2: Ausgabe des Programms U0904

Der String txa wird von Tastatur eingelesen. Die Funktion strlen() gibt die Länge des Strings als ganze Zahl an. Mit Hilfe der Funktion strcpy() wird dem String txb die Zeichenkette "Soft" zugewiesen. Mit Hilfe der Funktion strcat() wird dem String txb die Zeichenkette "ware" angehängt. Die Funktion strcmp() vergleicht zwei Strings Zeichen für Zeichen nach Ihrer Position im ASCII-Code und gibt ein Ergebnis zurück.

Hinweis: Zeichenketten müssen in doppelten Anführungszeichen geschrieben werden. Man muss bei Zeichenketten genau wie bei Feldern von Zahlen darauf achten, dass die Feldgrenzen nicht überschritten werden. Durch das Überschreiten von Feldgrenzen können andere Variablen zerstört (=überschrieben) werden.

Übung U0905:

Speichern Sie die Zeichenketten "Struk", "to" und "gramm" in drei Feldern. Lassen Sie sich die Zeichenketten und ihre Längen ausgeben. Fügen Sie sie anschließend zu der Zeichenkette "Struktogramm" zusammen (Datei U0905.c).

Übung U0906:

Speichern Sie die Zeichenkette "Programmiertechnik" in einem Feld der Größe 30. Setzen Sie im ersten Feld das Zeichen ‚\0' nacheinander an die Positionen 15, 12 und 5 (Datei U0906.c). Lassen Sie sich nach jedem Einsetzen jeweils anzeigen:

- alle Zeichen des Feldes einzeln mit %c
- die gesamte Zeichenkette mit %s
- die Länge der Zeichenkette

Übung U0907:

Ein Programm soll einen Satz einlesen und das Vorkommen des Buchstabens ‚e' ermitteln. Lassen Sie die Anzahl ausgeben und den prozentualen Anteil an allen Zeichen des Satzes (Datei U0907.c).

Übung U0908:

Ein Programm soll einen Satz einlesen und ihn ohne Leerzeichen wieder ausdrucken (Datei U0908.c).

Übung U0909:

In einem Programm soll ein dreibuchstabiges Wort (nur aus kleinen Buchstaben) erraten werden. Der erste Benutzer gibt das Wort ein, der zweite muss es herausfinden. Überprüfen Sie auf sinnvolle Eingaben. Nach jeder Eingabe bekommt der Benutzer die Information, ob er vor oder hinter dem Wort liegt (Datei U0909.c).

Übung U0910:

Schreiben Sie ein Programm, das einen Text einliest, alle darin enthaltenen Kleinbuchstaben (außer den Umlauten) in Großbuchstaben umwandelt und den gesamten Text wieder ausgibt (Datei U0910.c). Schreiben Sie dazu eine Funktion, die als Parameter ein Zeichen bekommt und als Rückgabewert ein Zeichen zurückliefert. Falls das übergebene Zeichen ein Kleinbuchstabe ist, soll der entsprechende Großbuchstabe zurückgeliefert werden, ansonsten soll das Zeichen unverändert zurückgeliefert werden.

10 Weitere Übungen

Alle Übungen sind in die Kategorien L (leicht), M (mittel) und S (schwer) eingeteilt.

Übung U1001 (M):

Einen Temperaturwert, der in Celsius gegeben ist, kann man mit folgender Formel in Fahrenheit umrechnen: F = C * 9 / 5 + 32. Schreiben Sie ein Programm, das eine Tabelle für 15 Celsius-Werte und die zugehörigen Fahrenheit-Werte herausgibt. Der größte und der kleinste Celsius-Wert dieser Tabelle sollen eingelesen werden. Die Werte dazwischen sollen in gleichmäßigem Abstand stehen (Datei U1001.c).

Übung U1002 (S):

Schreiben Sie ein Programm, das die Differenz zwischen zwei Datumsangaben des gleichen Jahres errechnet. Eingelesen werden Tag und Monat des ersten Datums und Tag und Monat des zweiten Datums. Ausgegeben wird die Differenz der beiden Daten in Tagen. Sorgen Sie dafür, dass nur sinnvolle Daten eingegeben werden können. Das betreffende Jahr ist kein Schaltjahr (Datei U1002.c).

Übung U1003 (M):

Schreiben Sie ein Programm, das die nachfolgende Bildschirmausgabe inkl. Rahmen erzeugt (Datei U1003.c). Die Elemente des Rahmens können Sie mithilfe der Zeichen 179, 192 bis 196 und 218 und dem Platzhalter %c erzeugen.

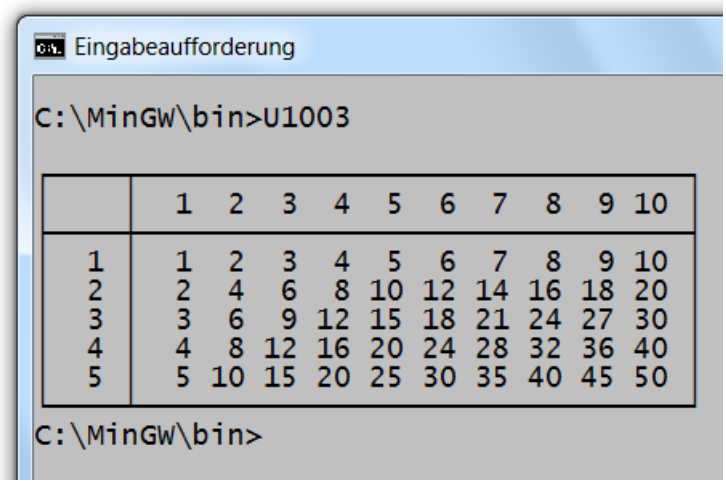


Abbildung 10.1: Ausgabe des Programms U1003

Übung U1004 (L):

Welche Ausgabe erzeugt folgendes Programm:

```
/* Datei U1004.c */
#include <stdio.h>
void main(void)
{
        int x,y;
        for (x=-3; x<2; x=x+1)
        {
             printf("\n");
             for (y=7; y>2; y=y-1)
                  printf("%4i ",2*x+y);
        }
}
```

Übung U1005 (M):

Mit Hilfe des folgenden Programms werden eingegebene Zahlen solange multipliziert, bis das Produkt die Zahl 300 überschreitet. Welche drei Fehler sind im Programm enthalten?

```
/* Datei U1005.c */
#include <stdio.h>
void main(void)
{
       int zahl, prod;
       while (prod < 300)
       {
            printf("\nZahl eingeben: ");
            scanf("%i",zahl);
            prod = prod * zahl;
            printf("Produkt: %i,prod);
       }
}
```

Übung U1006 (L):

Welche Ausgabe erzeugt folgendes Programm:

```
/* Datei U1006.c */
#include <stdio.h>
void main(void)
{
int i;
       for (i=5;i<25;i=i+2)
       {
            if (i<18)
            {
                 if (i<12)
                      printf("X");
                 else
                      printf("Y");
            }
            else
                 printf("Z");
       }
}
```

Übung U1007 (M):

Mit Hilfe des folgenden Programms werden zu den eingegebenen Zahlen die Kehrwerte ausgegeben (Kehrwert = 1 / Zahl). Welche drei Fehler sind im Programm enthalten?

```
/* Datei U1007.c */
#include <stdio.h>
float kw(double);

void main(void)
{
       double a;
       printf("\nZahl eingeben: ");
       scanf("%lf",&a);

       while (a!=0)
       {
            printf("Kehrwert von %lf ist %lf",a,kw(a));
            printf("\nZahl eingeben: ");
            scanf("%lf",&b);
       }
}

double kw(double x)
{
       y = 1/x;
       return(y);
}
```

Übung U1008 (S):

Schreiben Sie ein Programm, in dem maximal 10 double-Zahlen von Tastatur eingelesen werden, in einem Feld gespeichert werden und aufsteigend sortiert wieder ausgegeben werden (Datei U1008.c).

Benutzen Sie dazu das Sortierverfahren Bubble-Sort: Das Feld wird immer wieder durchlaufen, dabei werden jeweils zwei benachbarte Elemente miteinander verglichen und gegebenenfalls vertauscht. Das Sortierverfahren wird abgebrochen, sobald bei einem Durchlauf keine Vertauschung mehr stattgefunden hat.

11 Mehrdimensionale Felder

In C können auch mehrdimensionale Felder deklariert werden. Es werden dann mehrere Indizes benötigt, um die Position eines Elements in einem Feld anzugeben.

11.1 Aufbau

Die Anweisung "int a[5][8]" ermöglicht die Speicherung von insgesamt 40 int-Zahlen. Diese sind, wie in einer Tabelle, in 5 Zeilen und 8 Spalten angeordnet.

Man wählt ein einzelnes Element eines zweidimensionalen Feldes aus, indem man die Nummer seiner Zeile und seiner Spalte angibt. Mit der Anweisung "a[3][5] = 123" wird dem Element von a, das in der Zeile 3 und der Spalte 5 steht, der Wert 123 zugewiesen. Man nennt die Nummer der Zeile auch Zeilenindex, die Nummer der Spalte auch Spaltenindex. Die Indizes beginnen, wie bei eindimensionalen Feldern, bei 0.

Betrachten wir das folgende Programm:

```
/* Datei U1101.c */
#include <stdio.h>
void main(void)
{     int k, a[5][8];
```

```
        /* Jede Zeile hat 8 Elemente mit den Indizes 0 bis 7 */
        for (k=0;k<8;k++)
             a[0][k] = 3;      /* Zeile 0, Spalte k */

        /* Ausgabe */
        printf("Die erste Zeile lautet: ");
        for (k=0;k<8;k++)
             printf("%i ", a[0][k]);
}
```

Allen Elementen der ersten Zeile des oben angegebenen Feldes wird der Wert 3 zugewiesen. Anschließend wird diese erste Zeile wie folgt ausgegeben:

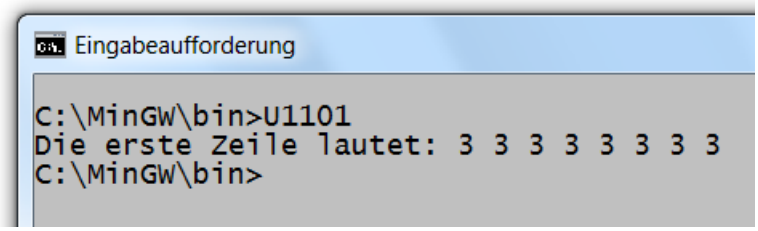


Abbildung 11.1: Ausgabe des Programms U1101

Übung U1102:

Schreiben Sie ein Programm, das den Elementen der ersten Zeile den Wert 6, den Elementen der zweiten Zeile den Wert 10, den Elementen der dritten Zeile den Wert 14 usw. zuweist. Anschließend sollen alle Zeilen des zweidimensionalen Feldes ausgegeben werden (Datei U1102.c).

Übung U1103:

Schreiben Sie ein Programm, dass die Zahlen des "Kleinen Einmaleins" in einem zweidimensionalen Feld der Größe 10 mal 10 erzeugt und anschließend ausgibt (Datei U1103.c).

Übung U1104:

Verändern Sie das Programm aus Übung U1103 so, dass die Elemente der vorletzten und der letzten Spalte des zweidimensionalen Feldes miteinander vertauscht werden. Zwei Zahlen werden mit Hilfe der folgenden Technik miteinander vertauscht:

- Speicherung der ersten Zahl in einer zusätzlichen Variablen
- Speicherung der zweiten Zahl in der ersten Zahl
- Speicherung der zusätzlichen Variablen in der zweiten Zahl

Das veränderte Feld soll anschließend ausgegeben werden (Datei U1104.c).

Übung U1105:

Statt der vorletzten und der letzten Spalte sollen nun zwei Zeilen miteinander vertauscht werden. Zu Beginn soll vom Programm nach den Nummern der beiden zu vertauschenden Zeilen gefragt werden. Es sollen nur sinnvolle Eingaben möglich sein (0 bis 9). Das veränderte Feld soll anschließend ausgegeben werden (Datei U1105.c).

11.2 Initialisierungen von Feldern

Ein- oder mehrdimensionale Felder können, genau wie einzelne Variable, schon bei der Deklaration mit Werten belegt werden. Dies geschieht in der Form von Zuweisungen, diese sind ausserhalb der Deklarationen nicht möglich. Im folgenden Programm sind einige Beispiele enthalten:

```
/* Datei U1106.c */
#include <stdio.h>

void main(void)
{
        int a[3] = {3,4,5};
        int b[5] = {2,4,6};
        double c[5] = {4.2,5.3,6.4,7.5};
        int d[] = {-1,5,9,22,4};

        int e[2][3] = {{3,4,5},{9,8,7}};
        int f[][3] = {{3,4},{9,8,7},{0},{4,8}};
}
```

Zur Initialisierung von Feldern werden die beiden geschweiften Klammern benötigt ({ und }).

- Feld a wird mit drei Elementen vollständig belegt
- Bei Feld b werden die Elemente 0, 1 und 2 belegt, die restlichen Elemente bekommen den Wert 0
- Bei Feld c werden die Elemente 0, 1, 2 und 3 belegt, das letzte Element bekommt den Wert 0.0
- Die Größe von Feld d wird erst mit der Initialierung festgelegt. Innerhalb der Klammern stehen 5 Zahlen, also hat das Feld 5 Elemente. Felder, bei denen die Größe nicht angegeben ist, müssen initialisiert werden.
- Bei der Initialisierung von zweidimensionalen Feldern (siehe Feld e) werden die geschweiften Klammern für jede Zeile benötigt
- Bei Feld f ist nur die Anzahl der Spalten bekannt, die Anzahl der Zeilen wird durch die Initialisierung festgelegt. Nicht belegte Elemente werden mit 0 besetzt. Die Anzahl der Spalten muss bekannt sein.

Übung U1106:

Lassen Sie alle Felder des obigen Programms mithilfe von geeigneten for-Schleifen wie folgt ausgeben:

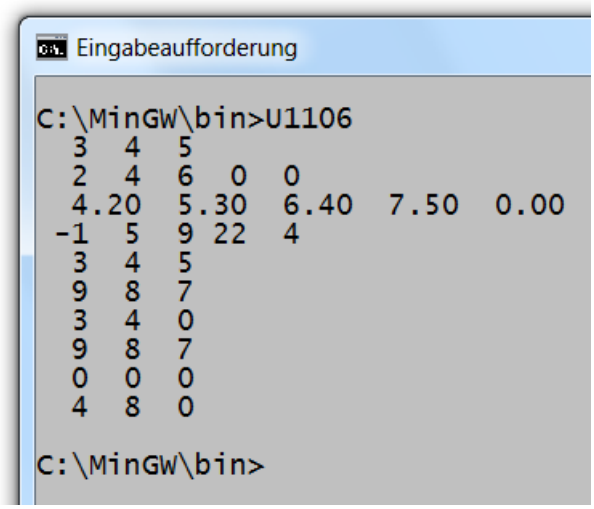


Abbildung 11.2: Ausgabe des Programms U1106

12 Felder von Zeichenketten

Eine einzelne Zeichenkette wird als eindimensionales Feld von Zeichenkonstanten gespeichert. Analog dazu wird ein Feld von Zeichenketten als zweidimensionales Feld von Zeichenkonstanten gespeichert. Der erste Index bezeichnet das Element des Feldes, der zweite Index den Buchstaben der Zeichenkette. Die Zeichenkettenfunktionen können entsprechend eingesetzt werden, siehe folgendes Beispiel:

```
/* Datei U1201.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char g[2][20];
        int i, vgl;
```

```
        printf("Geben Sie einen Text ein: ");
        gets(g[0]);
        strcpy(g[1],"Soft");
        for (i=0;i<2;i++)
             printf("\nText %i: %s, Laenge: %i",
                  i,g[i],strlen(g[i]));

        strcat(g[0],"_Ende");
        g[1][4] = 'w';
        g[1][5] = 'a';
        g[1][6] = 'r';
        g[1][7] = 'e';
        g[1][8] = '\0';
        for (i=0;i<2;i++)
             printf("\nText %i: %s, Laenge: %i",
                  i,g[i],strlen(g[i]));

        vgl = strcmp(g[0],g[1]);
        printf("\n%i\n",vgl);
        if (vgl < 0)
             printf("Text 0 im ASCII-Code vor Text 1");
        else if (vgl > 0)
             printf("Text 0 im ASCII-Code hinter Text 1");
        else
             printf("Text 0 = Text 1");

        printf("\n");
        for (i=0;i<40;i++)
             printf("%c",g[0][i]);

        printf("\n");
}
```

Die Ausgabe des Programms, abhängig von der Eingabe des Benutzers:

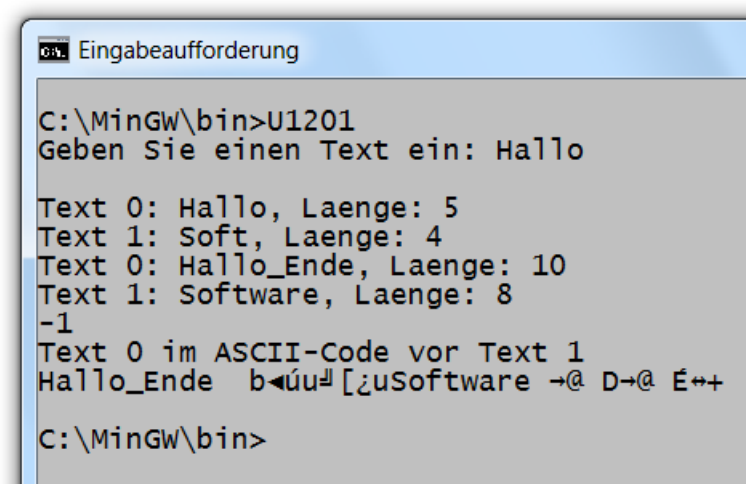


Abbildung 12.1: Ausgabe des Programms U1201

- Es wird ein Feld für zwei Zeichenketten der Länge 20 deklariert.
- In das erste Feldelement wird eine Zeichenkette von Tastatur eingelesen.
- Dem zweiten Feldelement wird die Zeichenkette "Soft" zugewiesen.
- Beide Zeichenketten werden mit Text und Länge ausgegeben.
- Dem ersten Feldelement wird die Zeichenkette "XXX" angehängt.
- Der zweiten Zeichenkette wird der Text "ware" in einzelnen Buchstaben angehängt, dabei muss die überschriebene binäre Null auch angehängt werden.
- Beide Zeichenketten werden wiederum mit Text und Länge ausgegeben.
- Die beiden Zeichenketten werden Zeichen für Zeichen nach Ihrer Position im ASCII-Code verglichen und das Ergebnis des Vergleiches wird ausgegeben.
- Wie andere zweidimensionale Felder werden auch diese Felder von Zeichenkonstanten zeilenweise hintereinander im Speicher abgelegt. Dies kann man anhand der letzten Ausgabe überprüfen.
- Da die Feldinhalte zeichenweise ausgegeben werden, werden auch die zufälligen Zeichen hinter dem regulären Ende der jeweiligen Zeichenkette ausgegeben. Dies können sichtbare oder unsichtbare Zeichen sein.

Übung U1202:

Erzeugen Sie ein Feld für vier Zeichenketten der Länge 20. Speichern Sie die Zeichenketten "Struk", "to" und "gramm" in den ersten drei Feldelementen. Fügen Sie sie anschließend zu der Zeichenkette "Struktogramm" im vierten Feldelement zusammen. Lassen Sie sich alle Zeichenketten und ihre Längen ausgeben (Datei U1202.c).

Übung U1203:

Schreiben Sie ein Programm (Datei U1203.c), das für ein eingelesenes Wort mit maximal 20 Buchstaben die folgende Bildschirmausgabe erzeugt (hier am Beispiel des Wortes "Berg"):

```
B
Be
Ber
Berg
Ber
Be
B
```

Übung U1204:

Schreiben Sie ein Programm, in dem drei Sätze eingelesen werden können (maximal 75 Zeichen pro Satz). In jedem Satz soll die Anzahl jedes der fünf Vokale, egal ob Klein- oder Großbuchstabe, ermittelt werden. Pro Satz sollen diese Anzahlen und der prozentuale Anteil der Vokale an allen Zeichen des Satzes ausgegeben werden. Außerdem soll die Anzahl jedes der fünf Vokale über alle drei Sätze ermittelt und ausgegeben werden (Datei U1204.c).

Übung U1205:

Schreiben Sie ein Programm, in dem fünf Namen eingegeben werden können (pro Name maximal 20 Zeichen). Diese Namen sollen anschließend mit Hilfe der Funktion strcmp() alphabetisch sortiert wieder ausgegeben werden (Datei U1205.c).

Benutzen Sie dazu das Sortierverfahren Bubble-Sort. Zur Erinnerung: Das Feld wird immer wieder durchlaufen, dabei werden jeweils zwei benachbarte Elemente miteinander verglichen und gegebenenfalls vertauscht. Das Sortierverfahren wird abgebrochen, sobald bei einem Durchlauf keine Vertauschung mehr stattgefunden hat.

Übung U1206:

Schreiben Sie ein Programm, in dem in einem zweidimensionalen Feld von Zeichenketten (!) der Sitzplan der Teilnehmer eines Kurses abgelegt ist. Anschließend sollen die Namen in jeder Reihe nach Länge sortiert werden, wiederum mit Bubble Sort, kürzester Name vorne. Der veränderte Sitzplan soll auf dem Bildschirm ausgegeben werden (Datei U1206.c).

13 Mehr zu Datentypen und Variablen

In diesem Kapitel lernen Sie weitere Datentypen kennen. Der Operator sizeof ermittelt die Speichergröße einer Variablen. Der Gültigkeitsbereich kann lokal oder global sein. Variablen können von einem zum anderen Datentyp umgewandelt werden.

13.1 Datentypen

Bisher haben wir die grundlegenden Datentypen char, int, float und double kennengelernt. Diese können teilweise durch Modifizierer genauer bestimmt werden. Als Modifizierer für die ganzzahligen Datentypen char und int gibt es: signed, unsigned, long und short. Einige C-Compiler unterstützen für double ebenfalls den Modifizierer long.

Im Folgenden eine Tabelle der möglichen Datentypen. Zusätzlich ist die Standard-Größe einer Variablen des jeweiligen Datentyps im Speicher und der Wertebereich angegeben. Bei nicht - ganzzahligen Datentypen ist es außerdem wichtig, die Genauigkeit zu kennen. Diese Angaben können je nach Hardware und Entwicklungsumgebung unterschiedlich sein.

Datentyp	Größe	Wertebereich	Genauigkeit	Platzhalter
char = signed char	1 Byte	-128 bis +127		%c
unsigned char	1 Byte	0 bis 255		%c
short int = signed short int	2 Byte	-32.768 bis +32.767		%i
unsigned short int	2 Byte	0 bis 65.535		%i
int = signed int = long int = signed long int	4 Byte	-2.147.483.648 bis +2.147.483.647		%i
unsigned int = unsigned long int	4 Byte	0 bis 4.294.967.295		%i
float	4 Byte	$3.4*10^{-38}$ bis $3.4*10^{+38}$	7 Stellen	%f
double	8 Byte	$1.7*10^{-308}$ bis $1.7*10^{+308}$	15 Stellen	%lf

13.2 Der Operator sizeof

Der Operator sizeof ermittelt die Größe eines Datenobjektes in Byte. Diese Größe kann sowohl für einen Datenyp als auch für eine Variable eines Datentyps verwendet werden. Er kann auch für zusammengesetzte Datentypen verwendet werden, wie sie später noch eingesetzt werden. Beispiel: Falls man die Variable a vom Typ int deklariert hat, liefert sowohl sizeof(int) als auch sizeof(a) den Wert 2.

Übung U1301:

Schreiben Sie ein Programm, in dem Sie sich mit Hilfe von sizeof für die folgenden Datentypen die Größe in Byte auf Ihrem PC angeben lassen: char, int, long int, float, double, long double (Datei U1301.c).

13.3 Lokale Variablen

Alle Variablen, die in den bisherigen Programmen verwendet wurden, waren lokale Variablen. Sie werden nur innerhalb der Funktion benutzt, in der sie auch deklariert werden. Die gleiche Variable kann mehrmals in einem Programm vorkommen und unterschiedliche Werte beinhalten. Man muss sich nicht merken, welche Variablen in anderen Funktionen bereits benutzt worden sind. Dies wird im folgenden Programm gezeigt:

```
/* Datei U1302.c */
#include <stdio.h>

void funk1(void)
{
       int x=20;
       printf("\n%i",x);
}

void main(void)
{
```

```
        int x;
        for (x=5;x<8;x++)
        {
             printf("\n%i",x);
             funk1();
        }
}
```

Die Ausgabe des Programms:

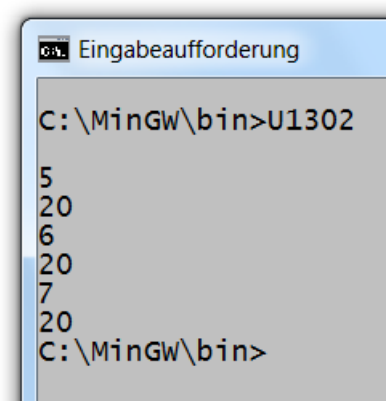


Abbildung 13.1: Ausgabe des Programms U1302

Der Wert der Variablen x ist in der Funktion bzw. im Hauptprogramm unterschiedlich.

13.4 Globale Variablen

Im Gegensatz zu lokalen Variablen können globale Variablen von allen Funktionen eines Programms benutzt werden. Globale Variablen werden außerhalb aller Funktionen deklariert, meist am Anfang eines Programms. Dies wird im folgenden Programm gezeigt:

```
/* Datei U1303.c */
#include <stdio.h>

int x;

void funk1(void)
{
       printf("\n%i",x);
}

void main(void)
{
       x=20;
       funk1();

       for (x=5;x<8;x++)
       {
            printf("\n%i",x);
            funk1();
       }
}
```

Die Ausgabe des Programms:

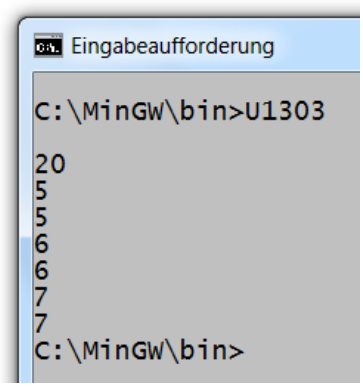


Abbildung 13.2: Ausgabe des Programms U1303

Ein Hinweis: Versuchen Sie möglichst, globale Variablen zu vermeiden. Die einzelnen Funktionen eines Programms sollten unabhängig voneinander arbeiten können, mit klar definierten Schnittstellen zwischen den Funktionen. Dies erhöht die Modularität Ihrer Programme und die Wiederverwendbarkeit der Funktionen.

Es gibt noch eine Besonderheit: Falls in einer Funktion eine lokale Variable mit dem gleichen Namen wie eine globale Variable deklariert wurde, so hat die lokale Variable innerhalb der Funktion Vorrang. Dies wird im folgenden Programm gezeigt:

```
/* Datei U1304.c */
#include <stdio.h>

int x;

void funk1(void)
{
        int x=10;
        printf("\n%i",x);
}

void main(void)
{
        x=20;
        funk1();

        for (x=5;x<8;x++)
        {
             printf("\n%i",x);
```

```
            funk1();
        }
}
```

Übung U1305:

Welche Ausgabe erzeugt das folgende Programm. Lösen Sie die Aufgabenstellung möglichst ohne PC:

```
/* Datei U1305.c */
#include <stdio.h>
void f1(int);

void f2(void)
{
        int z=5;
        printf("\nf2: %i",z);
        z++;
}

int z=12;

void f3(void)
{
        printf("\nf3: %i",z);
        z++;
}

void main(void)
{
        int z=17;
        printf("\nmain: %i",z);
        z++;
        f3();
        f2();
        f1(z);
}

void f1(int z)
{
        printf("\nf1: %i",z);
        z++;
}
```

13.5 Casts

Es gibt zwei Möglichkeiten, Variablen verschiedener Datentypen miteinander zu verrechnen bzw. umzuwandeln, implizit oder explizit.

- Die implizite Methode wendet man automatisch an, wenn in einem Ausdruck oder einer Anweisung Variablen verschiedener Datentypen stehen.
- Bei der expliziten Methode verwendet man sogenannte Casts. Ein Cast ist die Angabe des gewünschten Datentyps in Klammern vor dem Ausdruck, der umgewandelt werden soll.

Bei beiden Methoden sollte man die Vorgehensweise des Compilers beachten, damit kein unerwünschter Datenverlust auftritt. Einige häufige Operationen werden am folgenden Beispiel erläutert:

```
/* Datei U1306.c */
#include <stdio.h>

void main(void)
{
        int a=8, b=3, c;
        double d=0.4, e, f=0.3;

        /* Ganzzahlen miteinander verrechnen */
```

```
        c=a*100/b;              printf("\n1: %i",c);
        c=a/b*100;              printf("\n2: %i",c);

        /* Ganzzahlen in Fließkommazahlen umwandeln */
        e=a;                    printf("\n3: %lf",e);
        e=a/b;                  printf("\n4: %lf",e);
        e=(double)a/b;          printf("\n5: %lf",e);
        e=(double)(a/b);        printf("\n6: %lf",e);

        /* Fließkommazahlen in Ganzzahlen umwandeln */
        c=d;                    printf("\n7: %i",c);
        c=d/f;                  printf("\n8: %i",c);
        c=(int)d/f;             printf("\n9: %i",c);
        c=(int)(d/f);           printf("\n10: %i",c);

        /* Ganzzahlen und Fließkommazahlen miteinander verrechnen */
        c=a*d;                  printf("\n11: %i",c);
        e=a*d;                  printf("\n12: %lf",e);
}
```

- 1: Vorgehensweise von links nach rechts: Zuerst wird a mal 100 gerechnet, anschließend wird der Ausdruck durch b geteilt, die Nachkommastellen gehen erst dann verloren.
- 2: Zuerst wird a durch b geteilt, dabei gehen schon die Nachkommastellen verloren. Anschließend wird der Ausdruck mit 100 multipliziert, das Ergebnis ist wesentlich ungenauer als in Beispiel 1.
- 3: Implizite Umwandlung von Ganzzahl in Fließkommazahl, kein Problem.
- 4: Zuerst wird a durch b gerechnet, dabei gehen schon die Nachkommastellen verloren. Die implizite Umwandlung von Ganzzahl in Fließkommazahl geschieht "zu spät".
- 5: Vorgehensweise von links nach rechts: a wird mit Hilfe eines Casts explizit in eine Fließkommazahl umgewandelt, diese wird durch die Ganzzahl b geteilt, Ergebnis ist die richtige Fließkommazahl. Es tritt kein Verlust auf.
- 6: Zuerst wird a durch b gerechnet, dabei gehen schon die Nachkommastellen verloren. Die explizite Umwandlung von Ganzzahl in Fließkommazahl mit Hilfe eines Casts geschieht "zu spät".
- 7: Bei der Umwandlung von Fließkommazahlen mit Nachkommastellen in Ganzzahlen treten immer Verluste auf, die Ergebnisse sind jedoch unterschiedlich. Bei der impliziten Umwandlung von Fließkommazahl in Ganzzahl werden die Nachkommastellen abgeschnitten.
- 8: Zwei Fließkommazahlen werden dividiert, das Ergebnis ist die richtige Fließkommazahl. Bei der impliziten Umwandlung werden "nur einmal" die Nachkommastellen abgeschnitten.
- 9: Die erste Fließkommazahl wird explizit mit Hilfe eines Casts zur Ganzzahl gemacht, dabei gehen schon die Nachkommastellen verloren. Danach wird die Ganzzahl mit einer Fließkommazahl verrechnet, Ergebnis ist eine Fließkommazahl. Diese wird implizit in eine Ganzzahl umgewandelt, wiederum werden die Nachkommastellen abgeschnitten.

- 10: Zwei Fließkommazahlen werden dividiert, das Ergebnis ist die richtige Fließkommazahl. Die explizite Umwandlung ist überflüssig, da danach noch eine implizite Umwandlung stattfindet. Dabei werden "nur einmal" die Nachkommastellen abgeschnitten.
- 11: Eine Fließkommazahl und eine Ganzzahl werden zur richtigen Fließkommazahl verrechnet. Bei der impliziten Umwandlung werden "nur einmal" die Nachkommastellen abgeschnitten.
- 12: Eine Fließkommazahl und eine Ganzzahl werden zur richtigen Fließkommazahl verrechnet. Anschließend Zuweisung zu einer Fließkommazahl. Es tritt kein Verlust auf.

Die Ausgabe des Programms:

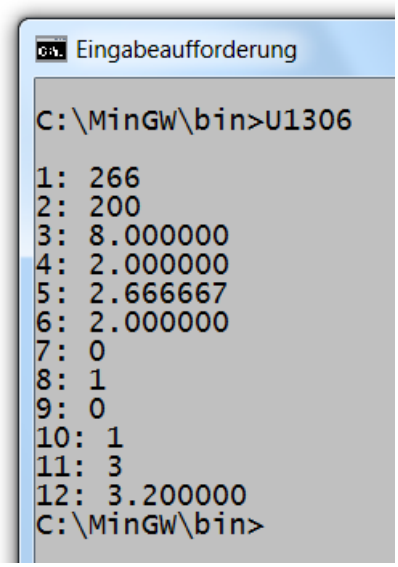


Abbildung 13.3: Ausgabe des Programms U1306

Übung U1307:

Welche Ausgabe erzeugt das folgende Programm. Lösen Sie die Aufgabenstellung möglichst ohne PC:

```
/* Datei U1307.c */
#include <stdio.h>
void main(void)
{
        int a=2, b=3, c;
        double d=2.5, e, f=5.2;
        e = d * b / (int)f;
        printf("\n%lf",e);
        c = d / (double)b * a;
        printf("\n%i",c);
        printf("\n%lf",e/c);
        printf("\n%i",(int)e/c);
}
```

14 Zeiger

Zunächst werden Stellenwertsysteme, Konstanten und Umwandlungsfunktionen vorgestellt. Anschließend wird das wichtige Thema Zeiger und Adressen erläutert.

14.1 Stellenwertsysteme

Ein Stellenwertsystem ist ein System zur Darstellung von Zahlen durch Ziffern, bei denen der Wert einer Ziffer von der Stelle abhängt, an welcher sie innerhalb der Zahl geschrieben ist. Das gebräuchlichste Stellenwertsystem ist das Dezimalsystem (Zahlen zur Basis 10). In der Informatik werden außerdem das Dualsystem (Basis 2) und das Hexadezimalsystem (Basis 16) eingesetzt, seltener das Oktalsystem (Basis 8).

Die benutzten Ziffern in den verschiedenen Systemen sind: Im Dualsystem 0 und 1, im Oktalsystem 0 bis 7, im Dezimalsystem 0 bis 9, im Hexadezimalsystem 0 bis 9 und A bis F. Die Buchstaben A bis F nehmen dabei die Dezimal - Werte von 10 bis 15 an.

Beispiele:

- Dezimal-Zahl 456: 4*102 + 5*101 + 6*100 = 400 + 50 + 6 = 456
- Dual-Zahl 11001: 1*24 + 1*23 + 0*22 + 0 * 21 + 1 * 20 = 16 + 8 + 0 + 0 + 1 = 25 (dezimal)
- Hexadezimal-Zahl 2A5F: 2*163 + 10*162 + 5*161 + 15*160 = 2*4096 + 10*256 + 5*16 + 15 = 10847 (dezimal)

14.2 Konstanten

In den bisherigen Programmen wurden neben Variablen auch Konstanten benötigt, z.B. zur Zuweisung von Werten und zu Berechnungen von Ausdrücken. Im folgenden eine Zusammenfassung der verschiedenen Konstantentypen mit ihren Schreibweisen.

Ganzzahl-Konstanten:

- 6815, Dezimal
- 0x1A9F, Hexadezimal (beginnend mit 0x)
- 015237, Oktal (beginnend mit 0)

Fließkomma-Konstanten:

- 2.4f, float
- 3., Wert 3,0
- -.4, Wert -0,4
- -24.0e-1, Wert -2,4 (Exponentialschreibweise)
- -04.E+2, Wert -400,0 (Exponentialschreibweise)

Die Konstanten, die einen Punkt enthalten, sind automatisch Fließkomma-Konstanten und haben den Typ double. Ausnahme: Falls ein f angehängt wird, sind sie vom Typ float.

Zeichen-Konstanten:

- ‚a', Zeichen klein a
- ‚\n', Zeichen "Neue Zeile"

Zeichenketten-Konstanten:

- "Erste Zeile\nNächste Zeile", String mit Steuerzeichen

14.3 Umwandlungsfunktionen atoi und itoa

Mit Hilfe der Umwandlungsfunktionen atoi() und itoa() kann man Zeichenketten in Zahlen verwandeln und umgekehrt.

Die Funktion atoi() aus der Bibliothek stdlib.h wandelt die Zeichen einer Zeichenkette in eine int-Zahl um, falls dies möglich ist. Aufruf-Beispiel: i = atoi(s).

- Falls die Zeichenkette mit Ziffern oder einem Vorzeichen beginnt, werden diese bis zur ersten Nicht-Ziffer ausgewertet und i enthält die entsprechende Zahl
- Falls die Zeichenkette mit Zeichen beginnt, enthält i die Zahl 0

Die Funktion itoa() aus der Bibliothek stdlib.h wandelt eine int-Zahl in eine Zeichenkette um. Beispiel für einen Aufruf: itoa (i,s,basis). Die int-Zahl i wird in die Zeichenkette s umgewandelt, in "basis" steht die Zahlenbasis für die Umwandlung. Falls man die int-Zahl z.B. im dualen System haben möchte, so muss man als Basis 2 angeben. Erlaubt sind als Basis Werte zwischen 2 und 32.

Betrachten wir das folgende Programm, in dem die Funktion atoi() eingesetzt wird:

```
/* Datei U1401.c */
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
       int i;
       char s[10];

       do
       {
            gets(s);                /* Zeichenkette einlesen */
```

```
            i = atoi(s);              /* Zeichenkette in Zahl umwandeln */
            printf("%i\n",i);         /* Zahl ausgeben */
        }
        while(i!=0);              /* Abbruch, falls ZK keine Zahl oder Zahl = 0 */
}
```

Die Ausgabe des Programms:

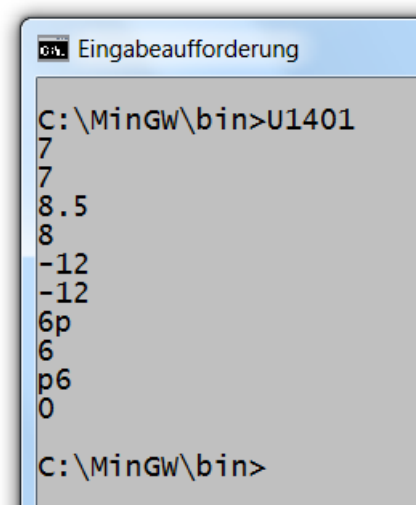


Abbildung 14.1: Ausgabe des Programms U1401

In diesem Programm wird solange eine Zeichenkette von Tastatur eingelesen, in eine int-Zahl umgewandelt und als Zahl ausgegeben wird, bis man auf eine Zeichenkette stößt, die nicht umgewandelt werden kann.

Übung U1402:

Schreiben Sie ein Programm, das die int-Zahlen von 1 bis 20 als 001, 002, 003, ... 019, 020 ausgibt. Dazu sollen die Zahlen in Zeichenketten verwandelt werden und mit Nullen aufgefüllt werden (Datei U1402.c).

Übung U1403:

Schreiben Sie unter Verwendung der Funktion itoa() ein Programm, das die int-Zahlen von 1 bis 20 als Tabelle in vier verschiedenen Darstellungen auf dem Bildschirm ausdruckt: dual (Basis 2), oktal (Basis 8), dezimal (Basis 10), hexadezimal (Basis 16). Versehen Sie die Tabelle mit einer Überschrift (Datei U1403.c).

14.4 Zeiger und Adressen

Variablen werden unter bestimmten Adressen im Speicher des Rechners abgelegt. Mit Hilfe von Zeigern (Pointern) kann man sich diese Adressen anzeigen lassen. Außerdem kann man über diese Adressen auch mit den zugehörigen Variablen arbeiten. Das folgende Beispielprogramm, verdeutlicht den Einsatz von Zeigern:

```
/* Datei U1404.c */
#include <stdio.h>
void main(void)
{
        float a=1.3f, b=2.5f;
        float c[10];
        float *fp1, *fp2;

        c[5]=23.87f;
        fp1=&a;
        b=*fp1;
        *fp1=0.0f;
        fp1=&c[5];
        printf("%8.4f %8.4f %8.4f\n",a,b,*fp1);

        fp2=fp1;
        b=*fp2+1;
        *fp1+=5;
        a=*fp1;
        printf("%8.4f %8.4f %8.4f\n",a,b,*fp1);
}
```

- Mit "float *fp1, *fp2" werden zwei Zeiger-Variablen vom Typ "Zeiger auf float-Variable" deklariert.
- Mit "fp1=&a" bekommt die Zeiger-Variable fp1 als Wert die Adresse von a zugewiesen. "&" nennt man den Adreßoperator. Man sagt auch: fp1 zeigt auf die Variable a.
- Mit "b=*fp1" bekommt die Variable b den Wert der Variablen, auf die der Zeiger fp1 zeigt, zugewiesen. "*" nennt man im Zusammenhang mit Zeigern den Inhaltsoperator. b hat nun also den Wert von a.
- Mit "fp1=&c[5]" bekommt die Zeiger-Variable fp1 eine neuen Wert, die Adresse des Feldelementes 5 des Arrays c.
- Mit "fp2=fp1" bekommt die Zeiger-Variable fp2 den gleichen Wert wie fp1, also die Adresse von c[5] zugewiesen. fp2 zeigt jetzt auch auf c[5].
- Mit "b=*fp2+1" bekommt die Variable b den Wert der Variablen, auf die der Zeiger fp2 zeigt, erhöht um 1, zugewiesen. b hat nun den Wert 24.87.
- Mit *fp1+=5" wird die Variable, auf die fp1 zeigt (das ist c[5]), um 5 erhöht. c[5] hat nun den Wert 28.87.
- Mit "a=*fp1" bekommt die Variable a den Wert der Variablen, auf die der Zeiger fp1 zeigt, zugewiesen. a hat nun also den Wert von c[5].

Die Ausgabe des Programms:

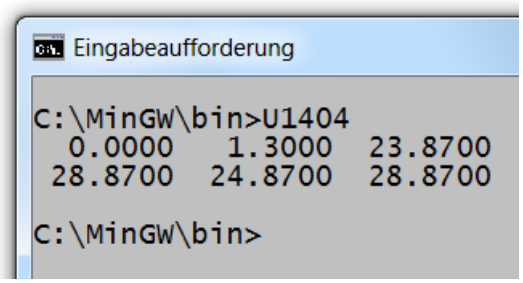


Abbildung 14.2: Ausgabe des Programms U1404

Übung U1405:

Schreiben Sie ein Programm (Datei U1405.c), in dem zwei Zeiger-Variablen vom Typ "Zeiger auf int-Variable" deklariert wird. Folgende Operationen sollen durchgeführt werden:

- einer int-Variablen soll ein Wert zugewiesen werden
- der ersten Zeiger-Variablen soll die Adresse dieser int-Variablen zugewiesen werden
- der Wert der Variablen soll, unter Verwendung des Zeigers, ausgegeben werden
- der Wert der Variablen soll, unter Verwendung des Zeigers, um 10 vermindert werden
- der zweiten Zeiger-Variablen soll der Wert der ersten Zeiger-Variablen zugewiesen werden
- der Wert, auf den die zweite Zeiger-Variable zeigt, soll ausgegeben werden

Übung U1406:

Erweitern Sie das Programm aus Übung U1405. Statt für eine einzelne int-Variable sollen die Operationen für alle Elemente eines Arrays aus 8 int-Variablen durchgeführt werden (Datei U1406.c).

14.5 Zeiger und Funktionen

Bisher konnten Funktionen nur einen einzigen Wert zurückliefern. Mit Hilfe von Zeigern kann man aus einer Funktion auch mehrere Werte zurückliefern. Das ist eine der wichtigsten Eigenschaften von Zeigern. Das folgende Beispielprogramm verdeutlicht diesen Zusammenhang:

```
/* Datei U1407.c */
#include <stdio.h>
void zf(int *, int *);

void main(void)
{
       int a=3,b=5,*pa,*pb;
       pa=&a;
       pb=&b;
       printf("%5i %5i\n",a,b);
       zf(pa,pb);
       printf("%5i %5i\n",a,b);
}

void zf(int *px, int *py)
{
```

```
        *px=*px+10;
        *py=*py+20;
}
```

- Den Zeiger-Variablen pa und pb werden die Adressen von a und b zugewiesen
- Vor Aufruf der Funktion zf() werden die Original - Werte ausgegeben
- Die Funktion zf() wird aufgerufen, Parameter sind die Adressen von a und b
- Innerhalb der Funktion haben jetzt px und py als Werte die Adressen von a und b
- Die Werte von a und b werden verändert, unter Verwendung der Zeiger px und py
- Nach Rückkehr in die Hauptfunktion main() werden die veränderten Werte ausgegeben

Die Ausgabe des Programms:

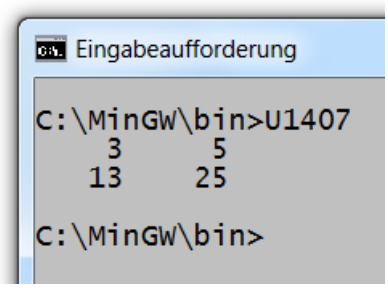


Abbildung 14.3: Ausgabe des Programms U1407

Übung U1408:

Schreiben Sie ein Programm (Datei U1408.c), in dem zwei double-Variablen miteinander getauscht werden. Der Tausch soll mit Hilfe einer Funktion swap() durchgeführt werden. Die Variablen sollen vor und nach dem Tausch im Hauptprogramm ausgegeben werden.

Übung U1409:

Schreiben Sie eine Funktion, die als Parameter eine double-Zahl bekommt und drei Zahlen an das aufrufende Programm zurück liefert: Zahl hoch 2, Zahl hoch 3, Zahl hoch 4. Die Funktion soll in einem geeigneten Hauptprogramm aufgerufen werden, alle Zahlen sollen ausgegeben werden (Datei U1409.c).

15 Anwendung von Zeigern

Zeiger und Felder sind in C sehr eng miteinander verwandt. Der Name eines Feldes ist gleichzeitig ein Zeiger auf das erste Element des Feldes. Zur Übergabe an und Rückgabe von Werten zu Funktionen spielen sie eine wichtige Rolle. Außerdem kann man mit Zeigern auch rechnen, dabei wird der Datentyp des Zeigers immer beachtet.

15.1 Aufbau

Der Aufbau soll an folgendem Programm verdeutlicht werden (Datei U1501.c):

```
/* Datei U1501.c */
#include <stdio.h>
void main(void)
{
        int a[5]={3,5,7,9,11}, *pa;
        printf("\n%i",a[0]);
        printf("\n%i",*a);
        printf("\n%i",a);
        pa=a;
        printf("\n%i",*pa);
        pa=pa+1;
```

```
        printf("\n%i",*pa);
        pa=a+3;
        printf("\n%i",*pa);
        /* a=a+1 */
}
```

- Ein Feld von int-Zahlen wird deklariert und initialisiert
- Das erste Element wird wie gewohnt ausgegeben
- Das erste Element wird über "Inhalt von Zeiger" ausgegeben
- Die Adresse des Feldes wird ausgegeben
- Der Zeiger pa bekommt die Adresse des Feldes, anschließend wird der Inhalt von pa ausgegeben, also das erste Feldelement
- Der Zeiger pa wird um eins erhöht. Da er vorher auf das erste Feldelement gezeigt hat, zeigt er nun auf das zweite Feldelement
- Zur Adresse des Feldes wird 3 addiert, damit zeigt pa auf das Element 3 des Feldes.
- Es ist allerdings nicht möglich, einen Zeiger zu verändern, der als Feldname dient.

Die Ausgabe des Programms:

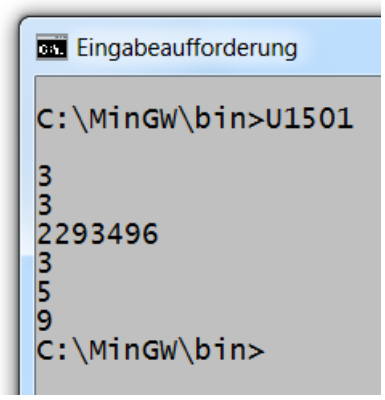


Abbildung 15.1: Ausgabe des Programms U1501

Übung U1502:

Schreiben Sie ein Programm, in dem ein Feld aus 4 double-Variablen deklariert und initialisiert wird. Die Adressen und Werte aller Elemente sollen mit Hilfe einer Schleife ausgegeben werden. Versuchen Sie, dazu mehrere Möglichkeiten zu finden, mit und ohne Zeiger (Datei U1502.c).

15.2 Zeiger und Funktionen

Übergibt man einer Funktion die Adresse eines Feldes, so kann innerhalb der Funktion auf die Feldelemente zugegriffen werden. Im folgenden Programm ist dies auf zwei Arten gelöst worden:

```
/* Datei U1503.c */
```

```
#include <stdio.h>
void f1(int *);
void f2(int *);

void main(void)
{
        int a[4]={1,8,6,2};
        f1(a);
        f2(a);
}

void f1(int *x)
{
        int i;
        for (i=0;i<4;i++)
             printf("%i ",x[i]);
}
void f2(int x[4])
{
        int i;
        for (i=0;i<4;i++)
             printf("%i ",x[i]);
}
```

Die Ausgabe des Programms:

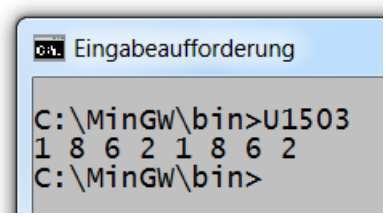


Abbildung 15.2: Ausgabe des Programms U1503

Innerhalb der Funktion f1() wird x als Zeiger auf eine int-Variable benutzt. Diese int-Variable kann gleichzeitig das erste Element eines Feldes sein. Daher können auch die restlichen Elemente des Feldes erreicht werden.

Innerhalb der Funktion f2() wird das Feld x unmittelbar mit seiner Feldgröße deklariert. Auf die ursprünglich deklarierte Größe des Feldes muss geachtet werden.

Übung U1504:

Schreiben Sie ein Programm, das mehrere Funktionen beinhaltet:

- Eine Funktion, in der die Summe, der Mittelwert und das Maximum der Elemente eines übergebenen Feldes berechnet und zurückgeliefert werden.
- Eine Funktion, die die Elemente eines Feldes in umgekehrter Reihenfolge in einem zweiten Feld speichert.
- Eine Funktion, die die Elemente eines Feldes nach Größe sortiert, größtes Element zuerst.

Diese Funktionen sollen für ein eindimensionales Feld von double-Zahlen ausgeführt werden. Den Funktionen sollen u.a. die Adresse und die Größe des Feldes übergeben werden. Im Hauptprogramm sollen alle benutzten Variablen und Felder ausgegeben werden (Datei U1504.c).

Übung U1505:

Schreiben Sie eine Funktion, der zwei Parameter übergeben werden: die Adresse eines Feldes von Zeichen (nennt man auch: Zeiger auf eine Zeichenkette) und ein einzelnes Zeichen. Innerhalb der Funktion soll festgestellt werden, wie oft das Zeichen in der Zeichenkette vorkommt. Diese Zahl soll als Rückgabewert der Funktion zurückgeliefert werden und im Hauptprogramm kommentiert ausgegeben werden (Datei U1505.c).

15.3 Funktionen und mehrdimensionale Felder

Bei der Übergabe der Adresse eines mehrdimensionalen Feldes sollte man in der Parameterliste der Funktion die Feldgröße direkt deklarieren. Dies soll am folgenden Programm gezeigt werden:

```
/* Datei U1506.c */
#include <stdio.h>
void ausgeb(int a[2][2]);

void main(void)
{
       int a[2][2]={{3,4},{8,9}};
       ausgeb(a);
}

void ausgeb(int a[2][2])
{
       int i, j;
       for (i=0;i<2;i++)
            for (j=0;j<2;j++)
                 printf("%3i",a[i][j]);
}
```

Die Ausgabe des Programms:

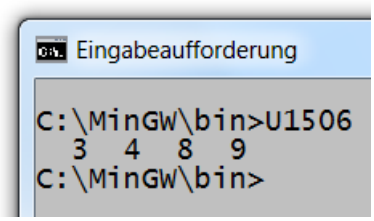


Abbildung 15.3: Ausgabe des Programms U1506

Übung U1507:

Schreiben Sie eine Funktion, der drei Parameter übergeben werden: die Adresse eines Feldes von Zeichenketten, ein einzelnes Zeichen und die Adresse eines int-Feldes. Innerhalb der Funktion soll festgestellt werden, wie oft das Zeichen in den einzelnen Zeichenketten jeweils vorkommt. Diese Zahlen sollen in dem int-Feld festgehalten und an das Hauptprogramm zurückgeliefert werden. Dort sollen sie kommentiert ausgegeben werden (Datei U1507.c).

Übung U1508:

Schreiben Sie eine Funktion, der zwei Zeichenketten übergeben werden. Innerhalb der Funktion soll festgestellt werden, ob die zweite Zeichenkette innerhalb der ersten Zeichenkette vorkommt. Falls dies der Fall ist, soll diese Position als Rückgabewert der Funktion zurückgeliefert werden. Falls dies nicht der Fall ist, soll der Wert -1 zurückgeliefert werden. Im Hauptprogramm soll das Ergebnis kommentiert ausgegeben werden (Datei U1508.c).

Übung U1509:

Schreiben Sie eine Funktion, der zwei Zeichenketten übergeben werden. Innerhalb der Funktion sollen die Zeichen der ersten Zeichenkette in umgekehrter Reihenfolge in der zweiten Zeichenkette gespeichert werden. Im Hauptprogramm sollen beide Zeichenketten ausgegeben werden (Datei U1509.c).

Übung U1510:

Schreiben Sie eine Funktion, der ein zweidimensionales Feld von double-Zahlen übergeben wird. Innerhalb der Funktion soll für jede Zeile des zweidimensionalen Feldes der Mittelwert berechnet werden. Diese Mittelwerte sollen an das Hauptprogramm übergeben werden und dort kommentiert ausgegeben werden (Datei U1510.c).

16 Dateizugriff

Bei der Ein- und Ausgabe von Daten in Dateien muss man wissen, welcher Dateityp vorliegt und welche Zugriffsart man verwenden kann. Man kann zwischen folgenden Zugriffsarten unterscheiden:

- Sequentieller Zugriff: Diese Möglichkeit hat man bei einer Datei, deren einzelne Zeilen unterschiedlich lang sind und jeweils mit einem Zeilenumbruch beendet werden. Ihr Inhalt kann mit einem ASCII-Editor bearbeitet werden. Sie werden rein sequentiell gelesen bzw. geschrieben. Es ist nicht möglich, auf eine bestimmte Zeile direkt zuzugreifen, da man nicht weiß, wie lang die Vorgängerzeilen sind.
- Wahlfreier Zugriff: Diese Möglichkeit hat man bei einer Datei, die größtenteils gleich lange Datensätze beinhaltet. Es können Zeilenumbrüche existieren, müssen aber nicht. Die Länge und Struktur eines Datensatzes sollte bekannt sein oder innerhalb der Datei an einer vereinbarten Stelle stehen. Sie können direkt gelesen bzw. verändert werden, da man den Ort jedes Datensatzes berechnen kann.
- Binärer Zugriff: Diese Zugriffsmöglichkeit hat man bei jeder Datei. Man arbeitet mit den reinen Byte-Folgen, diese können mit Hilfe eines darauf angepassten C-Programms gelesen oder verändert werden. Allerdings kann dies zur Folge haben, dass die Dateien nicht mehr mit den zugehörigen Anwendungsprogrammen gelesen werden können. Beispiel: Man überschreibt in einer Access-Datenbank die Stelle, an der die Anzahl der Datensätze einer bestimmten Tabelle steht. Dies kann dazu führen, dass diese Tabelle oder mehrere Tabellen zerstört werden.

Es gibt Mischformen zwischen den genannten Typen. Ohne Kenntnis der Struktur einer Datei ist es nicht möglich, sie korrekt zu bearbeiten. Man kann die Struktur unter Umständen mit Hilfe eines Hex-Editors ermitteln. Dieser zeigt den Inhalt im Klartext und gleichzeitig die Byte-Folgen in hexadezimaler Form.

16.1 Sequentielles Lesen einzelner Zeichen

Das folgende Programm öffnet eine Textdatei. Anschließend werden die ersten fünf Zeichen gelesen und die Datei wird wieder geschlossen.

```
/* Datei U1601.c */
#include <stdio.h>
void main(void)
{
        FILE *dp;
        int ch, i;
        dp = fopen("U1601.TXT","r");

        if (dp != NULL)
        {
             printf("\nDatei offen\n");
             for (i=1; i<=5; i++)
             {
                  ch = fgetc(dp);
                  printf("%c",ch);
             }
             fclose(dp);
             printf("\nDatei geschlossen\n");
        }
        else
             printf("\nDatei nicht offen");
}
```

FILE *dp; Es wird ein Zeiger auf eine Datei deklariert. Dieser Zeiger wird zur Bearbeitung der Datei benötigt.

dp = fopen("U1601.TXT","r"); Es wird versucht, die Datei U1601.TXT im gleichen Verzeichnis im Read-Modus (zum Lesen) zu öffnen. Falls dieser Versuch erfolgreich war, wird dem Dateizeiger die Adresse des ersten Byte der Datei zugewiesen. Der Versuch kann fehlschlagen, falls diese Datei nicht in diesem Verzeichnis existiert.

if (dp != NULL) bedeutet: "falls der Dateizeiger auf etwas zeigt". Diese Prüfung muss stattfinden, da weitere Befehle bei nicht erfolgreicher Datei-Öffnung zu Fehlern führen können.

ch = fgetc(dp); Es wird ein Zeichen aus der Datei in die Variable ch gelesen. Diese Funktion ist ähnlich zu getch(). Der Dateizeiger zeigt danach auf das nächste Byte.

fclose(dp); Die Datei wird geschlossen. Diese Funktion macht nur Sinn, wenn die Datei auch geöffnet wurde. Sie sollte aufgerufen werden; man sollte sich nicht darauf verlassen, dass die Datei nach Beendigung des C-Programms automatisch geschlossen wird. Unter Umständen ist die Datei ansonsten gesperrt und kann von keinem Programm bearbeitet werden.

Der Inhalt der Datei, die gelesen werden soll:

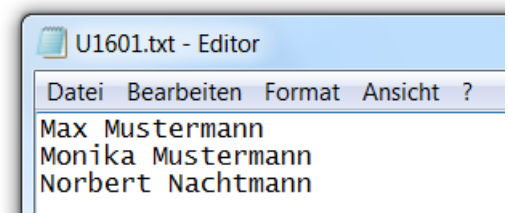


Abbildung 16.1: Inhalt der Textdatei U1601.TXT

Die Ausgabe des Programms:

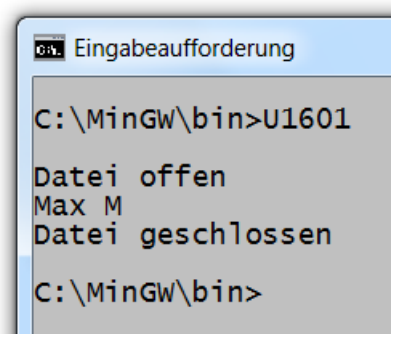


Abbildung 16.2: Ausgabe des Programms U1601

16.2 Sequentielles Lesen aller Zeichen einer Datei

Das folgende Programm liest alle Zeilen aus einer einfachen Textdatei. Sie hat den gleichen Inhalt wie im vorherigen Abschnitt, siehe Abbildung 16.1.

```
/* Datei U1602.c */
#include <stdio.h>
void main(void)
{
        FILE *dp;
        char z[200];
        int i;
```

```
        dp = fopen("U1602.TXT","r");

        if (dp != NULL)
        {
             while(feof(dp) == 0)
             {
                  fgets(z, 200, dp);
                  printf("%s",z);
             }
             fclose(dp);
        }
        else
             printf("\nDatei nicht geöffnet");
}
```

while(feof(dp) == 0) Stellt fest, ob das Ende der Datei erreicht wurde. Falls dies der Fall ist, gibt feof einen Wert ungleich 0 zurück. Falls das Ende der Datei noch nicht erreicht wurde, gibt feof den Wert 0 zurück.

fgets(z, 200, dp); Die Funktion fgets() liest eine Zeichenfolge aus der Datei und speichert sie in z. Die Zeichen werden von der aktuellen Position in der Datei bis einschließlich zum ersten "Neue-Zeile"-Zeichen (\n), bis zum Ende des Datenflusses oder bis die Anzahl der gelesenen Zeichen gleich 199 ist, gelesen, je nachdem was zuerst der Fall ist. An die gelesenen Zeichen wird ein Nullzeichen (\0) angefügt. Das "Neue-Zeile"-Zeichen wird, falls es gelesen wurde, in die Zeichenfolge eingeschlossen.

Zusatz: Je nachdem, ob die Datei am Ende der letzten sinnvollen Zeile oder am Anfang der darauffolgenden Zeile beendet wurde, wird das Dateiende zu unterschiedlichen Zeitpunkten erkannt. Das vorliegende Programm setzt die erste Variante voraus.

Die Ausgabe des Programms:

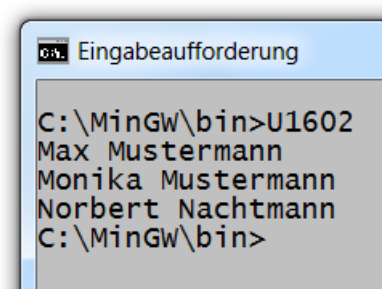


Abbildung 16.3: Ausgabe des Programms U1602

Übung U1603:

Schreiben Sie ein Programm, das feststellt, wie oft ein bestimmtes Zeichen in den einzelnen Zeilen einer Datei und innerhalb der gesamten Datei vorkommt (Datei U1603.c). Außerdem soll die Anzahl der Zeilen festgestellt werden. Die Ausgabe soll wie folgt aussehen:

```
Zeile: 0    Anzahl: 2
Zeile: 1    Anzahl: 0
Zeile: 2    Anzahl: 4
Zeile: 3    Anzahl: 4
...
Zeilen insgesamt: 14
Anzahl insgesamt: 24
```

16.3 Sequentielles Schreiben einer Datei

Das folgende Programm schreibt zwei Zeilen in eine Textdatei.

```
/* Datei U1604.c */
#include <stdio.h>
void main(void)
{
        FILE *dp;
```

```
        int nr = 12;
        char zkette[20] = {"Ausgabezeile"};
        dp = fopen("U1604.TXT","w");

        if (dp != NULL)
        {
             fprintf(dp,"%i %s\n", nr, zkette);
             fprintf(dp,"Zweite Zeile");
             fclose(dp);
        }
        else
             printf("Datei nicht geöffnet\n");
}
```

dp = fopen("U1604.TXT","w"); Es wird versucht, die Datei U1604.TXT im gleichen Verzeichnis im Write-Modus (zum Schreiben) zu öffnen. Falls dieser Versuch erfolgreich war, wird dem Dateizeiger die Adresse des ersten Byte der Datei zugewiesen. Der Versuch kann fehlschlagen, falls das Laufwerk schreibgeschützt oder voll ist. Falls die Datei vorher existiert hat, wird sie ohne Warnung (!) gelöscht und überschrieben.

fprintf(dp,"%i %s\n", nr, zkette); Es wird die angegebene Ausgabe in die Datei geschrieben. Die Funktion fprintf() arbeitet ähnlich wie die Funktion printf().

Die neu erzeugte Datei sieht wie folgt aus:

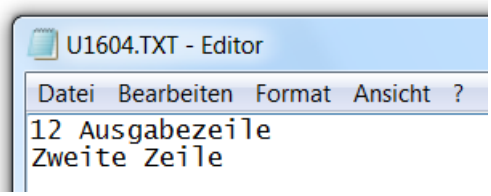


Abbildung 16.4: Inhalt der Textdatei U1604.TXT

Übung U1605:

Schreiben Sie ein Programm, in dem alle Zeilen einer Eingabe-Datei gelesen werden. Falls innerhalb der ersten drei Zeichen einer Zeile das Zeichen ‚#' vorkommt, soll diese Zeile in Ausgabedatei A geschrieben werden. Falls dies nicht der Fall ist, soll diese Zeile in Ausgabedatei B geschrieben werden (Datei U1605.c).

16.4 Wahlfreier Lese-Zugriff

Das folgende Programm liest einzelne Zeichen an beliebigen Stellen einer Datei.

```
/* Datei U1606.c */
#include <stdio.h>
void main(void)
{
       char x;
       FILE *dpe;
       dpe = fopen("U1606.TXT","r");

       if (dpe != NULL)
       {
            printf("Position: %i\n",ftell(dpe));
            fseek(dpe, 8, SEEK_SET);
            printf("Position: %i\n",ftell(dpe));
            x = fgetc(dpe);
            printf("Zeichen: %c\n",x);
            printf("Position: %i\n",ftell(dpe));
            fseek(dpe, -3, SEEK_CUR);
            printf("Position: %i\n",ftell(dpe));
            x = fgetc(dpe);
            printf("Zeichen: %c\n",x);
            printf("Position: %i\n",ftell(dpe));
            fseek(dpe, -3, SEEK_END);
            printf("Position: %i\n",ftell(dpe));
            x = fgetc(dpe);
            printf("Zeichen: %c\n",x);
            printf("Position: %i\n",ftell(dpe));

            fclose(dpe);
       }
       else
       {
            printf("Datei nicht gefunden");
       }
}
```

fseek(dpe, 8, SEEK_SET); Innerhalb der Datei wird der Dateizeiger 8 Byte weiter gesetzt, ausgehend von der Position SEEK_SET (= Dateianfang).

fseek(dpe, -3, SEEK_CUR); Innerhalb der Datei wird der Dateizeiger 3 Byte zurück gesetzt, ausgehend von der Position SEEK_CUR (= aktuelle Position).

fseek(dpe, -3, SEEK_END); Innerhalb der Datei wird der Dateizeiger 3 Byte zurück gesetzt, ausgehend von der Position SEEK_END (= Dateiende).

ftell(dpe) Es wird die aktuelle Position innerhalb der Datei zurückgegeben. Dies kann man benutzen, um später zur gleichen Stelle zurückzukehren.

Falls die Datei U1606.TXT wie folgt aussieht:

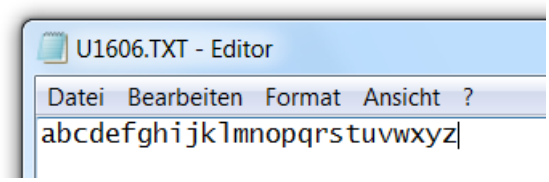


Abbildung 16.6: Inhalt der Textdatei U1606.TXT

dann erzeugt das Programm die nachfolgende Ausgabe:

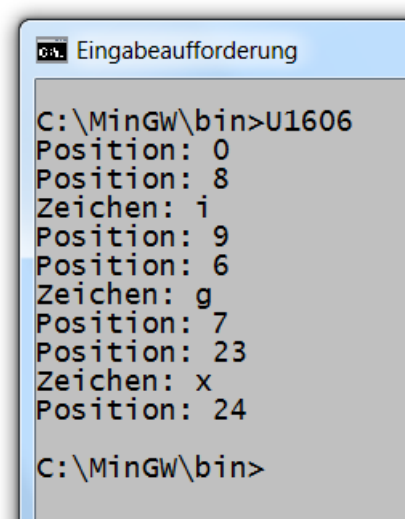


Abbildung 16.6: Ausgabe des Programms U1606

16.5 Formatiertes Schreiben in eine Datei

Das folgende Programm schreibt einzelne Datensätze gleicher Länge in eine Datei. Eine solche Datei ist Voraussetzung für den wahlfreien Lese- und Schreibzugriff.

```
/* Datei U1607.c */
#include <stdio.h>
void main(void)
{
       FILE *dpe;
       dpe = fopen("U1607.TXT","w");
       if (dpe != NULL)
       {
            fprintf(dpe, "%3i %-13s%9.2lf\n", 15, "Mueller", 37.80);
            fprintf(dpe, "%3i %-13s%9.2lf\n", 25, "Maier", 47.80);
            fprintf(dpe, "%3i %-13s%9.2lf\n", 33, "Momper", 87.80);
            fprintf(dpe, "%3i %-13s%9.2lf\n",  8, "Mustermann", 137.80);
            fprintf(dpe, "%3i %-13s%9.2lf\n", 23, "Moeller", 237.80);
            fclose(dpe);
       }
       else
       {
            printf("Datei nicht gefunden");
       }
}
```

fprintf(...) An der aktuellen Position des Dateizeigers wird ein formatierter Datensatz ausgegeben.

Die neu erzeugte Datei sieht wie folgt aus:

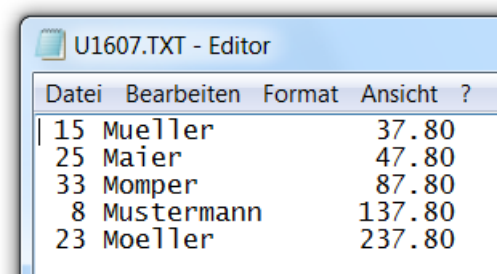


Abbildung 16.7: Inhalt der Textdatei U1607.TXT

16.6 Wahlfreier Lese- und Schreibzugriff

Das folgende Programm liest und verändert einen einzelnen Datensatz mitten in einer Datei.

```
/* Datei U1608.c */
#include <stdio.h>
void main(void)
{
        int nummer, dsize=28;
        char name[20];
        double betrag;
        FILE *dpe;

        dpe = fopen("U1607.txt","r+");

        if (dpe != NULL)
        {
             fseek(dpe,2*dsize,SEEK_SET);
             fscanf(dpe,"%i",&nummer);
             fscanf(dpe,"%s",name);
             fscanf(dpe,"%lf",&betrag);
             printf("%i\n%s\n%lf\n", nummer, name, betrag);

             fseek(dpe,2*dsize,SEEK_SET);
             fprintf(dpe, "%3i %-13s%9.2lf", 11, "Merkens", 35.90);

             fseek(dpe,2*dsize,SEEK_SET);
             fscanf(dpe,"%i",&nummer);
```

```
            fscanf(dpe,"%s",name);
            fscanf(dpe,"%lf",&betrag);
            printf("%i\n%s\n%lf\n",nummer, name, betrag);
            fclose(dpe);
        }
        else
        {
            printf("Datei nicht gefunden");
        }
}
```

dpe = fopen("U1607.txt","r+"); Es wird versucht, die Datei U1607.TXT im gleichen Verzeichnis im Lese- und Schreibmodus zu öffnen.

fscanf(dpe,"%i",&nummer); An der aktuellen Position des Dateizeigers wird eine int-Variable eingelesen.

fscanf(dpe,"%s",name); An der aktuellen Position des Dateizeigers wird eine Zeichenkette eingelesen.

fscanf(dpe,"%lf",&betrag); An der aktuellen Position des Dateizeigers wird eine double-Variable eingelesen.

Die Datei wurde verändert, beachten Sie den dritten Datensatz:

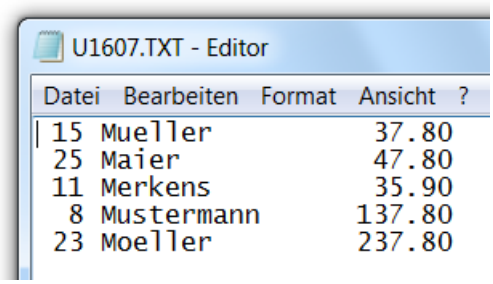


Abbildung 16.8: Inhalt der Textdatei U1607.TXT, nach der Änderung

Übung U1609:

Schreiben Sie ein Programm (Datei U1609.c), das die Bearbeitung des nachfolgend beschriebenen Dateityps mit Hilfe eines Benutzermenüs ermöglicht.

- Jeder Datensatz besteht nur aus einer Zeichenkette mit 20 Zeichen, beendet mit einem Zeilenumbruch. Dies erleichtert die Kontrolle und die Ein- und Ausgabe per Editor, muss allerdings beim Versetzen des Dateizeigers innerhalb der Datei berücksichtigt werden.
- In der ersten Zeile steht statt eines Datensatzes die Anzahl der Datensätze. Ab der zweiten Zeile stehen die eigentlichen Datensätze.
- Lesen und Ausgeben aller Datensätze.
- Das Verändern eines beliebigen Datensatzes: Der Benutzer gibt eine Nummer ein, das Programm prüft, ob diese Nummer existiert, fragt nach dem neuen Inhalt, trägt den neuen Inhalt ein und gibt alle Datensätze aus.
- Das Hinzufügen eines Datensatzes: Das Programm fragt nach dem Inhalt des neuen Datensatzes, erhöht die Anzahl der Datensätze um 1, trägt den neuen Inhalt ein und gibt alle Datensätze aus.

Das Benutzermenü soll wie folgt aussehen:

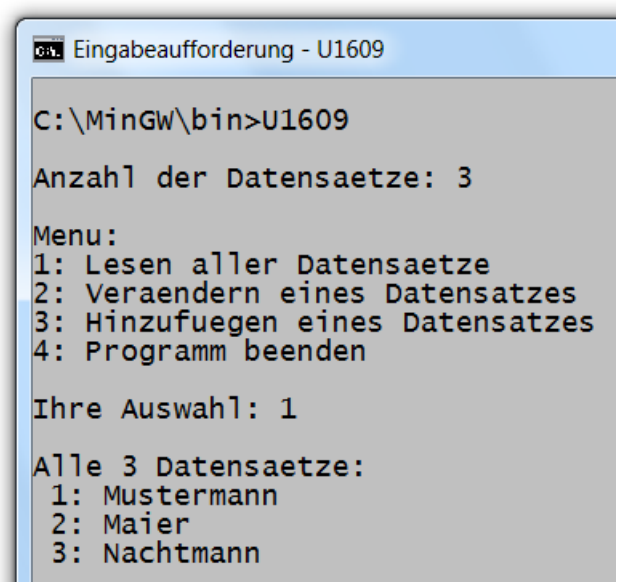


Abbildung 16.9: Benutzermenü des Programms U1609

Die zugehörige Datei wird nachfolgend im Notepad++ inklusive Sonderzeichen dargestellt:

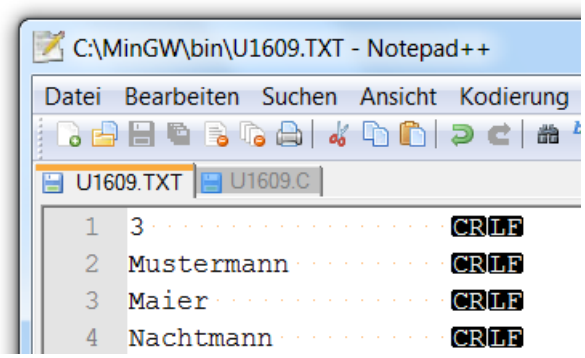


Abbildung 16.10: Inhalt der Textdatei U1609.TXT

17 Strukturen

Strukturen in C dienen dazu, mehrere Variablen mit unterschiedlichen Datentypen zu einer Einheit mit einem einzigen Namen zusammenzufassen.

17.1 Aufbau

Mit der Anweisung:

```
struct adr
{
        char strasse[30];
        int hausnr;
        long int plz;
        char ort[20];
};
```

wird der Struktur-Datentyp adr definiert. Objekte von diesem Typ besitzen vier Komponenten: zwei Zeichenketten und zwei int-Variablen.

Im nachfolgenden Programm wird der o.a. Struktur-Datentyp benutzt. Von diesem Typ wird eine Variable definiert. Allen Komponenten dieser Variable werden Werte zugewiesen. Anschließend werden diese Werte ausgegeben.

```
/* Datei U1701.c */
#include <stdio.h>
#include <string.h>
void main(void)
{
       /* Definition des Strukturtyps adr */
       struct adr
       {
            char strasse[30];
            int hausnr;
            long int plz;
            char ort[20];
       };

       /* Deklaration einer Variablen dieses Strukturtyps */
       struct adr adra;

       /* Zuweisung der Komponenten-Inhalte */
       strcpy(adra.strasse,"Peterstrasse");
       adra.hausnr = 36;
       adra.plz = 52074;
       strcpy(adra.ort,"Aachen");

       printf("\n%s ",adra.strasse);
       printf("%i ",adra.hausnr);
       printf("in %li ",adra.plz);
       printf("%s",adra.ort);

       /* Das geht leider nur, falls alle Komponenten vom gleichen */
       /* Datentyp sind: printf("\n%s %i in %li %s",adra);         */
}
```

Die Ausgabe des Programms:

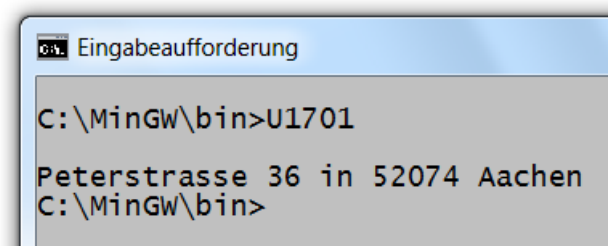


Abbildung 17.1: Ausgabe des Programms U1701

17.2 Zuweisung von struct-Variablen

Die Zuweisung adrb = adra weist allen Komponenten einer struct-Variablen die entsprechenden Werte der anderen struct-Variablen zu, sofern sie vom gleichen Typ ist.

Übung U1702:

Schreiben Sie ein Programm, in dem die Inhalte von zwei struct-Variablen des o.a. Typs mit Hilfe einer dritten Variablen vom gleichen Datentyp miteinander getauscht werden (Datei U1702.c). Zur Kontrolle sollen sie vor und nach dem Tauschvorgang ausgegeben werden.

17.3 Verschachtelte Strukturen

Die Komponente einer Struktur kann wiederum eine Struktur sein. Mit der Anweisung

```
struct mitarbeiter
{
        char vorname[20];
        char nachname[20];
        struct adr adrc;
        float gehalt;
} mita, mitb;
```

wird der Struktur-Datentyp mitarbeiter definiert. Objekte von diesem Typ besitzen vier Komponenten: zwei Zeichenketten, eine struct-Variable vom Typ adr und eine float-Variable. Die Struktur adr sollte vorher bekannt sein. Außerdem werden gleichzeitig zwei Variablen dieses Typs definiert.

Nachfolgend ein Programm, in dem der oben angegebene verschachtelte Struktur-Datentyp definiert wird. Von diesem Typ wird eine Variable definiert. Allen Komponenten dieser Variablen werden Werte zugewiesen. Anschließend werden sie ausgegeben.

```
/* Datei U1703.c */

#include <stdio.h>
#include <string.h>
void main(void)
{
        /* Definition des Unter-Strukturtyps adr */
        struct adr
        {
             char strasse[30];
             int hausnr;
             long int plz;
             char ort[20];
        };

        /* Definition des Ober-Strukturtyps mitarbeiter, gleichzeitig: */
        /* Deklaration einer Variablen dieses Ober-Strukturtyps          */
        struct mitarbeiter
        {
             char vorname[20];
             char nachname[20];
             struct adr adra;
```

```
        float gehalt;
    } mita;

    /* Zuweisung der Komponenten-Inhalte der Ober-Struktur */
    strcpy(mita.vorname,"Stefan");
    strcpy(mita.nachname,"Maier");
    mita.gehalt=5000;

    /* Zuweisung der Komponenten-Inhalte der Unter-Struktur */
    strcpy(mita.adra.strasse,"Peterstrasse");
    mita.adra.hausnr = 36;
    mita.adra.plz = 52074;
    strcpy(mita.adra.ort,"Aachen");

    printf("\n%s ",mita.vorname);
    printf("%s, ",mita.nachname);
    printf("\n%s ",mita.adra.strasse);
    printf("%i ",mita.adra.hausnr);
    printf("in %li ",mita.adra.plz);
    printf("%s",mita.adra.ort);
    printf("\nGehalt %.0f Euro",mita.gehalt);
}
```

Die Ausgabe dieses Programms:

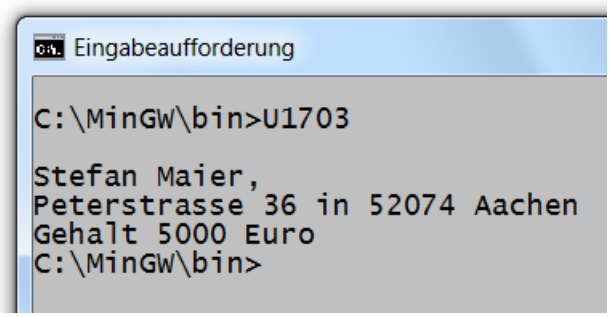


Abbildung 17.2: Ausgabe des Programms U1703

17.4 Felder von Strukturen

Wie bei den elementaren Datentypen kann man auch bei Struktur-Datentypen Felder definieren. Die Elemente eines solchen Feldes sind Variablen des Struktur-Datentyps.

Mit der Anweisung struct mitarbeiter personal[10] wird ein Feld von 10 Elementen des Datentyps mitarbeiter definiert. Mit personal[5].vorname kann man auf die Komponente vorname des Feld-Elementes 5 zugreifen.

Übung U1704:

Schreiben Sie ein Programm (Datei U1704.c), in dem den 10 Feldelementen des o.a. Feldes:

- bei der Komponente hausnr die Werte 50, 55, 60, 65 usw. zugewiesen werden
- bei der Komponente gehalt des o.a. Feldes die Werte 3500, 3520, 3540, 3560 usw. zugewiesen werden.
- diese beiden Komponenten eines Feldelementes gemeinsam ausgegeben werden

17.5 Zeiger auf Strukturen

Wie bei den elementaren Datentypen kann man auch bei Struktur-Datentypen Zeiger auf diese Typen definieren. Mit der Anweisung struct mitarbeiter *pa wird ein Zeiger mit dem Namen pa definiert. Dieser Zeiger zeigt auf Variablen vom Typ struct mitarbeiter. Mit der Anweisung pa = &mita wird der Zeiger-Variablen pa die Adresse der Variablen mita vom Struktur-Datentyp mitarbeiter zugewiesen.

Mit der Anweisung (*pa).gehalt = 4200 wird anschließend der Komponente gehalt der Variablen mita der Wert 4200 zugewiesen. Wichtig sind dabei die Klammern um den Namen des Zeigers. Als Alternative zu (*pa).gehalt kann man auch die Schreibweise pa->gehalt verwenden.

Übung U1705:

Schreiben Sie das Programm aus Übung U1702 um, dabei sollen (Datei U1705.c):

- die Daten der ersten Variablen in einer Funktion von Tastatur eingelesen werden
- die Daten der zweiten Variablen in einer Funktion zugewiesen werden
- der Tausch der Inhalte der beiden struct-Variablen in einer eigenen Funktion realisiert werden
- beide Schreibweisen für Komponenten von Strukturen eingesetzt werden

Übung U1706:

Wir betrachten eine räumliche Temperaturverteilung (Datei U1706.c). 5 Punkte in einem Raum sind durch ihre x-, y-, und z-Koordinaten sowie ihre Temperatur gekennzeichnet. Schreiben Sie ein Programm, in dem eine geeignete Struktur für dieses Problem definiert wird und ein geeignetes Feld von Struktur-Variablen deklariert wird. Außerdem sollen drei Funktionen geschrieben und aufgerufen werden:

- eine Funktion zur Eingabe der Daten der fünf Punkte
- eine Funktion zur Sortierung des Feldes nach der Temperatur
- eine Funktion zur Ausgabe der Daten der fünf Punkte

18 Dynamische Speicherverwaltung

In den bisherigen Programmen wurden Größe und Anzahl der Variablen bzw. der Felder bereits zum Zeitpunkt der Programmerstellung durch Deklarationen festgelegt. Diese Methode nennt man statische Speicherverwaltung. Es kann aber Aufgabenstellungen geben, die zur Lösung Felder benötigen, deren benötigte Größe erst während des Programmlaufes festgestellt wird.

Dies kann man dadurch lösen, dass man sehr große statische Felder vereinbart, deren Größe in jedem Fall ausreicht. Das führt allerdings zur Speicherplatzverschwendung und möglicherweise dazu, dass ein Programm gar nicht erst gestartet werden kann, da auf dem benutzten Rechner nicht genügend Speicher zur Verfügung steht.

Eine andere Lösung ist die dynamische Speicherverwaltung. Der benötigte Speicherplatz wird erst während des Programmlaufes angefordert und nach erfolgter Benutzung wieder freigegeben.

18.1 Funktion malloc

In C gibt es mehrere Funktionen zur dynamischen Speicherverwaltung, als wichtigste die Funktion malloc() aus der Bibliothek stdlib.h. Es folgt ein Programm mit der Anwendung von malloc():

```
/* Datei U1801.c */

#include <stdio.h>
#include <stdlib.h>
void main(void)
{
       /* Zeiger soll auf Beginn des angeforderten Speicherplatzes zeigen */
       int *zahl;

       /* Speicherplatz-Anforderung für 15 int-Zahlen */
       zahl = (int *) malloc(15 * sizeof(int));

       /* Speicherplatz-Anforderung erfolgreich ? */
       if (zahl != NULL)
            printf("Speicherplatz-Anforderung erfolgreich!\n");
       else
            printf("Speicherplatz-Anforderung nicht erfolgreich!\n");
}
```

Dabei bedeuten:

- malloc(15 * sizeof(int)): die Reservierung eines Bereiches für 15 int-Zahlen, Rückgabewert von malloc ist ein void-Zeiger auf diesen Bereich
- (int *): Umwandlung des void-Zeigers in einen int-Zeiger
- zahl: die Variable für diesen int-Zeiger

Mit Hilfe des Zeigers "zahl" kann man auf den resevierten Bereich zugreifen. Falls der Zeiger den Wert NULL hat, so war die Reservierung nicht erfolgreich.

Die Ausgabe des Programms, abhängig von der Eingabe des Benutzers:

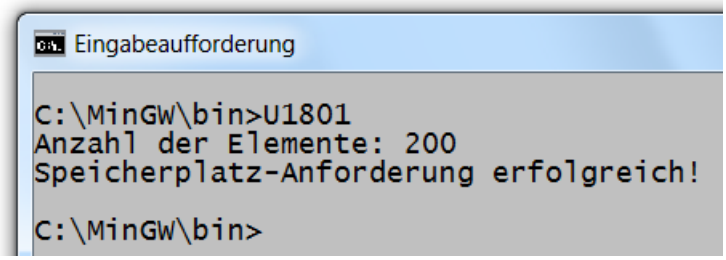


Abbildung 18.1: Ausgabe des Programms U1801

Übung U1802:

Erweitern Sie das Programm aus Übung U1801, indem Sie den reservierten Speicherplatz mit Zahlenwerten Ihrer Wahl belegen. Zur Erinnerung: der Zeiger "zahl" zeigt auf den Anfang dieses Bereiches und ist vom Typ Zeiger auf int. Lassen Sie sich den Inhalt des Bereiches vor und nach der Belegung ausgeben (Datei U1802.c).

18.2 Funktion free

Zur Freigabe des reservierten Speicherbereiches benötigt man die Funktion free(). Der Befehl free(zahl) gibt den Speicherbereich, auf den "zahl" zeigt, wieder frei zur späteren Benutzung für andere Programmteile. Man sollte einen reservierten Speicherbereich spätestens zum Programmende wieder freigeben.

Übung U1803:

Im folgenden Programm (Datei U1803.c) soll:

- dynamisch Speicherplatz für 5 double-Zahlen reserviert werden (Zeiger "zahl")
- der Speicherplatz mit Werten Ihrer Wahl belegt werden
- die Zahlen ausgegeben werden
- der reservierte Speicherplatz wieder freigegeben werden

18.3 Funktion calloc

Als Alternative zur Funktion malloc() kann die Funktion calloc() verwendet werden. Der Aufruf sieht etwas anders aus, außerdem wird der Bereich mit Nullen initialisiert. Beispiel für einen Aufruf:

```
zahl = (int *) calloc(10, sizeof(int);
```

Übung U1804:

Schreiben Sie das Programm aus Übung U1803 unter Verwendung von calloc statt malloc (Datei U1804.c). Die Größe des zu reservierenden Bereiches soll von Tastatur eingelesen werden. Lassen Sie sich wiederum den Inhalt des Bereiches vor und nach der Belegung ausgeben.

18.4 Funktion realloc

Wird während der Programmausführung festgestellt, dass der reservierte Bereich zu groß oder zu klein war, so kann er mit der Funktion realloc() verändert werden. Bei einer Vergrößerung bleiben die Daten im ursprünglichen Bereich erhalten. Bei einer Verkleinerung können die Daten im nicht mehr reservierten Bereich verloren gehen. Beispiel für einen Aufruf:

```
zahl = (int *) realloc(zahl, 15*sizeof(int));
```

Mit Hilfe dieses Aufrufes wird der bisher existierende Bereich, auf den der Zeiger "zahl" zeigt, auf die Größe von 15 int-Zahlen verändert.

Übung U1805:

Erweitern Sie das Programm aus Übung U1804, indem Sie während des Programmablaufes den reservierten Bereich einmal vergrößern lassen (Datei U1805.c). Lassen Sie sich jeweils die Größe und den Inhalt des Bereiches vor und nach den Belegungen ausgeben.

Übung U1806:

Schreiben Sie ein Programm, in dem dynamisch Speicherplatz für eine bestimmte Anzahl Objekte des nachfolgend angegebenen Strukturdatentyps belegt werden. Die Anzahl der der Objekte und die Daten der Objekte sollen von Datei eingelesen und auf dem Bildschirm ausgegeben werden (Datei U1806.c). Benutzen Sie zum Einlesen von Datei die Funktion fscanf(). Sie arbeitet ähnlich wie scanf(), zusätzlich mus als erster Parameter der Dateizeiger angegeben werden.

```
struct adr
{
       char strasse[30];
       int hausnr;
       long int plz;
       char ort[20];
};
```

19 Software

In diesem Kapitel werden Download, Installation und Benutzung der Software beschreiben, die für das Entwickeln von C-Programmen unter Windows oder Ubuntu Linux notwendig sind.

19.1 Download und Installation unter Windows

Ein häufig genutzter, frei verfügbarer C-Compiler für Windows ist MinGW. Die Abkürzung steht für Minimalist GNU for Windows. Die Installationsdatei ist zurzeit (Dezember 2011) mingw-get-inst-20111118. Sie hat eine Größe von ca. 0,6 MB und kann von der folgenden Website geladen werden:

http://sourceforge.net/projects/mingw/files/Installer/mingw-get-inst

Nach dem Download wird die Installationsdatei aufgerufen. Es bietet sich die Möglichkeit zur Installation verschiedener Compiler. Zumindest der C-Compiler muss installiert werden, weitere werden für diesen Kurs auch nicht benötigt. Nach der Auswahl des Compilers werden die gewünschten Elemente geladen und installiert. Das Standardverzeichnis ist C:\MinGW, die installierte Größe ca. 130 MB. Es wird kein Startmenüeintrag erzeugt (und auch keiner benötigt).

19.2 Benutzung unter Windows

Sie können Ihre C-Programme mit einem normalen Texteditor schreiben, wie er bereits mit Windows geliefert wird, zum Beispiel über Start / Programme / Zubehör / Editor. Alternativ können Sie auch Notepad++ nutzen. Dieser kleine (ca. 5,3 MB), frei verfügbare Editor bietet Syntax-Highlighting. Das bedeutet, dass die Sprachelemente farblich unterschiedlich dargestellt werden, dies hilft beim Programmieren. Notepad++ kann von der folgenden Website geladen werden: http://www.notepad-plus-plus.org

Schreiben Sie das nachfolgende Programm im Editor. Die Inhalte werden im eigentlichen Kurs erläutert:

```
#include <stdio.h>
void main(void)
{
       printf("Hallo Welt\n");
}
```

Speichern Sie die Datei unter dem Namen hallo.c im Verzeichnis C:\MinGW\bin. C-Programme werden in Dateien gespeichert, die als Endung den Kleinbuchstaben c haben. Rufen Sie die Eingabeaufforderung auf, über Start / Programme / Zubehör / Eingabeaufforderung. Wechseln Sie dort in das richtige Verzeichnis, mithilfe des folgenden Befehls:

```
cd \MinGW\bin
```

Beauftragen Sie den C-Compiler, Ihr C-Programm zu übersetzen, mit:

```
gcc hallo.c -o hallo
```

Es wird die Datei hallo.exe erzeugt. Rufen Sie sie auf mit:

```
hallo
```

Die Ausgabe Ihres Programms erscheint auf dem Bildschirm:

```
Hallo Welt
```

Sollten Sie Syntax-Fehler in Ihrem Programm haben, so erscheinen nach dem Übersetzen Fehlermeldungen. Diese müssen Sie erst beseitigen und erneut übersetzen, bevor Sie das fertige Programm aufrufen können. Anhand der Zeilennummer bei der Meldung können Sie erkennen, an welcher Stelle im Programm der jeweilige Fehler bemerkt wurde. Ab dieser Zeile aufwärts können Sie nach der Ursache des Fehlers suchen.

Hinweis: Falls Sie in der Eingabeaufforderung die Pfeiltaste nach oben betätigen, erscheinen die zuletzt eingegebenen Befehle. Dies spart Arbeit beim wiederholten Eingeben.

19.3 Download und Installation unter Ubuntu Linux

Unter dem (im Dezember 2011) aktuellen Ubuntu Linux 11.10 ist der GNU C-Compiler bereits installiert. Sollte dies nicht der Fall sein, so können Sie ihn von der folgenden Website aus installieren:

http://wiki.ubuntuusers.de/GCC

19.4 Benutzung unter Ubuntu Linux

Sie können Ihre C-Programme unter Ubuntu Linux mit dem Standard-Texteditor gedit schreiben. Er bietet Syntax-Highlighting. Das bedeutet, dass die Sprachelemente farblich unterschiedlich dargestellt werden, dies hilft beim Programmieren. C-Programme werden in Dateien gespeichert, die als Endung den Kleinbuchstaben c haben. In einem Terminal können Sie gedit wie folgt aufrufen:

```
gedit hallo.c
```

Schreiben Sie das nachfolgende Programm in gedit. Die Inhalte werden im eigentlichen Kurs erläutert:

```
#include <stdio.h>
void main(void)
{
       printf("Hallo Welt\n");
}
```

Speichern Sie die Datei und verlassen Sie gedit. Beauftragen Sie den C-Compiler, Ihr C-Programm zu übersetzen, mit:

```
gcc hallo.c -o hallo
```

Es wird die ausführbare Datei hallo erzeugt. Rufen Sie sie auf mit:

```
./hallo
```

Die Ausgabe Ihres Programms erscheint auf dem Bildschirm:

```
Hallo Welt
```

Sollten Sie Syntax-Fehler in Ihrem Programm haben, so erscheinen nach dem Übersetzen Fehlermeldungen. Diese müssen Sie erst beseitigen und erneut übersetzen, bevor Sie das fertige Programm aufrufen können. Anhand der Zeilennummer bei der Meldung können Sie erkennen, an welcher Stelle im Programm der jeweilige Fehler bemerkt wurde. Ab dieser Zeile aufwärts können Sie nach der Ursache des Fehlers suchen.

Hinweis: Falls Sie im Terminal die Pfeiltaste nach oben betätigen, erscheinen die zuletzt eingegebenen Befehle. Dies spart Arbeit beim wiederholten Eingeben.

20 Lösungen

20.1 Zu Kapitel 1

Lösung zu Übung U0102:

```
/* Datei U0102.c */
#include <stdio.h>
void main(void)
{
        /* Nach jeder Zeile folgt ein Zeilensprung mit \n */
        printf("Dieser Text\n");
        printf("besteht\n");
        printf("aus\n");
        printf("vier Zeilen\n");
}
```

Lösung zu Übung U0103:

```
/* Datei U0103.c */
#include <stdio.h>
void main(void)
```

```
{
       printf("Das Ergebnis\n");
       printf("von zwölf mal fünf\n");

       /* Platzhalter %i zur Ausgabe einer ganzen Zahl */
       printf("ist %i\n",12*5);
}
```

Lösung zu Übung U0104:

```
/* Datei U0104.c */
#include <stdio.h>
void main(void)
{
       printf("Einige Rechnungen:\n");
       printf("121+17=%i\n",121+17);
       printf("439-34=%i\n",439-34);
       printf("324+34-55=%i\n",324+34-55);

       /* Zwei Platzhalter für zwei Ergebnisse */
       printf("24+16/2=%i\n24+16/4=%i\n",24+16/2,24+16/4);

       /* Achtung: Innerhalb einer Zeichenkette darf kein Zeilenumbruch */
       /* erfolgen. Dies ist hier nur aus Platzgründen geschehen  */
       printf("Teil 1: 37-5*3=%i\nTeil 2: 137-7*4=%i
            \nTeil 3: 28+49/7=%i\n",37-5*3,137-7*4,28+49/7);

       /* Bei gleicher Priorität wird von links nach rechts ausgewertet */
       /* Daher muss im letzten Beispiel die Klammer gesetzt werden  */
       printf("48/3*2=%i\n",48/3*2);
       printf("48/(3*2)=%i\n",48/(3*2));
}
```

20.2 Zu Kapitel 2

Lösung zu Übung U0204:

```
/* Datei U0204.c */
#include <stdio.h>
void main(void)
{
       /* Deklaration der Variablen */
       float celsius;
```

```
        /* Eingabe der gewünschten Daten */
        printf("\nBitte Grad Celsius eingeben: ");
        scanf("%f",&celsius);

        /* Berechnung und Ausgabe des Ergebnisses */
        printf("\n%.1f Grad Celsius entsprechen ",celsius);
        printf("%.1f Grad Fahrenheit",celsius*9/5+32);
}
```

Lösung zu Übung U0205:

```
/* Datei U0205.c */
#include <stdio.h>
void main(void)
{
        /* Gleiche Reihenfolge aber umgekehrte Formel im Vergleich zu 2.1 */
        float fahr;
        printf("\nBitte Grad Fahrenheit eingeben: ");
        scanf("%f",&fahr);
        printf("\n%.1f Grad Fahrenheit entsprechen ",fahr);
        printf("%.1f Grad Celsius",(fahr-32)*5/9);
}
```

Lösung zu Übung U0206:

```
/* Datei U0206.c */
#include <stdio.h>
void main(void)
{
        /* Beim Ergebnis wird der Zahlenbereich der int-Zahlen    */
        /* überschritten. Damit kein falsches Ergebnis angezeigt wird, */
        /* kann man z.B. den Zahlenbereich der Eingabedaten vergrößern */
        float sek, min, std, tag;

        /* Alle benötigten Daten einlesen */
        printf("\nSekunden pro Minute: ");
        scanf("%f",&sek);
        printf("\nMinuten pro Stunde:  ");
        scanf("%f",&min);
        printf("\nStunden pro Tag    :  ");
        scanf("%f",&std);
        printf("\nTage               :  ");
        scanf("%f",&tag);
```

```
        /* Berechnung und Ausgabe des Ergebnisses */
        printf("\nGesamtzahl der Sekunden: %.0f", sek*min*std*tag);
}
```

Lösung zu Übung U0207:

```
/* Datei U0207.c */
#include <stdio.h>
void main(void)
{
        /* Zur Berechnung des Benzinverbrauches pro 100 Km werden zwei */
        /* Eingabedaten benötigt, eingelesen und gespeichert           */
        float bv, anz_km;

        printf("\nBenzinverbrauch: ");
        scanf("%f",&bv);
        printf("\ngefahrene Km   : ");
        scanf("%f",&anz_km);

        /* Berechnung und Ausgabe des Ergebnisses */
        printf("\nBenzinverbrauch / 100 Km: %.1f", bv / anz_km * 100);
}
```

Lösung zu Übung U0210:

```
/* Datei U0210.c */
#include <stdio.h>
void main(void)
{
       /* double-Variablen sind genauer und haben einen größeren */
       /* Zahlenbereich */
       double celsius;

       printf("\nBitte Grad Celsius eingeben: ");
       scanf("%lf",&celsius);

       /* Berechnung und Ausgabe des Ergebnisses mit 10 Nachkommastellen */
       printf("\n%.10lf Grad Celsius entsprechen ",celsius);
       printf("%.10lf Grad Fahrenheit",celsius*9/5+32);
}
```

Lösung zu Übung U0211:

```
/* Datei U0211.c */
#include <stdio.h>
void main(void)
{
       float sek, min, std, tag;

       /* Bekannte Werte werden zugewiesen, unbekannte Werte eingelesen */
       sek = 60;
       min = 60;
       std = 24;
       printf("\nTage: ");
       scanf("%f",&tag);

       /* Berechnung und Ausgabe des Ergebnisses */
       printf("\nGesamtzahl der Sekunden: %.0f", sek*min*std*tag);
}
```

Lösung zu Übung U0212:

```
/* Datei U0212.c */
#include <stdio.h>
void main(void)
{
```

```
        /* Die Anzahl der Tage der zwölf Monate müssen gespeichert werden */
        int jan, feb, mae, apr, mai, jun, jul, aug, sep, okt, nov, dez;

        /* Es sind vier Ergebnisse gefordert */
        int q1sum, q2sum, q3sum, q4sum;

        /* Zuweisung der Eingabedaten */
        jan = 31; feb = 28; mae = 31; apr = 30; mai = 31; jun = 30;
        jul = 31; aug = 31; sep = 30; okt = 31; nov = 30; dez = 31;

        /* Berechnung der Ergebnisse */
        q1sum = jan + feb + mae;
        q2sum = apr + mai + jun;
        q3sum = jul + aug + sep;
        q4sum = okt + nov + dez;

        /* Ausgabe der Ergebnisse */
        printf("\nSumme der Tage des 1. Quartals: %i", q1sum);
        printf("\nSumme der Tage des 2. Quartals: %i", q2sum);
        printf("\nSumme der Tage des 3. Quartals: %i", q3sum);
        printf("\nSumme der Tage des 4. Quartals: %i", q4sum);
}
```

20.3 Zu Kapitel 3

Lösung zu Übung U0302:

```
/* Datei U0302.c */
#include <stdio.h>
void main(void)
{
        /* Anfangswert und Endwert der Schleife sollen eingelesen werden */
        /* Außerdem wird eine Schleifenvariable benötigt      */
        int x, anf, ende;

        /* Gewünschte Anfangs- und Endwerte eingeben */
        printf("\nAnfang eingeben: ");
        scanf("%i",&anf);
        printf("\nEnde eingeben: ");
        scanf("%i",&ende);

        /* x durchläuft die Werte von Anfangswert bis einschließlich */
        /* Endwert in Schritten von 1     */
```

```
        for (x=anf; x<=ende; x=x+1)
        {
             printf("Zahl: ");
             /* Jeder Wert von x wird ausgegeben */
             printf("%i\n",x);
        }
}
```

Lösung zu Übung U0303:

```
/* Datei U0303.c */
#include <stdio.h>
void main(void)
{
        /* Während des Schleifendurchlaufes sollen alle Zahlen summiert */
        /* werden. Dies erfordert eine Variable für die Summe und eine  */
        /* Vorbesetzung mit 0 */
        int x, anf, ende, summe=0;

        /* Gewünschte Anfangs- und Endwerte eingeben */
        printf("\nAnfang eingeben: ");     scanf("%i",&anf);
        printf("\nEnde eingeben: ");       scanf("%i",&ende);
```

```
        /* Durchlauf aller Zahlen mit Summierung */
        for (x=anf; x<=ende; x=x+1)
        {
             printf("Zahl: ");
             printf("%i\n",x);
             /* Neuer Wert für die Summe ist alter Wert + aktueller Wert */
             /* Beim ersten Mal (x=anf) ist der alte Wert gleich 0      */
             summe = summe + x;
        }

        /* Nach der vollständigen Summierung Ausgabe des Ergebnisses */
        printf("\nSumme: %i",summe);
}
```

Lösung zu Übung U0304:

```
/* Datei U0304.c */
#include <stdio.h>
void main(void)
{
        /* Zusätzlich soll der Mittelwert ermittelt werden */
        float x, anf, ende, summe=0, mw;

        /* Gewünschte Anfangs- und Endwerte eingeben */
        printf("\nAnfang eingeben: ");
        scanf("%f",&anf);
        printf("\nEnde eingeben: ");
        scanf("%f",&ende);

        /* Durchlauf aller Zahlen mit Summierung */
        for (x=anf; x<=ende; x=x+1)
        {
             printf("Zahl: ");
             printf("%.0f\n",x);
             summe = summe + x;
        }

        /* Mittelwert = Summe der Zahlen / Anzahl der Zahlen           */
        /* Anzahl der Zahlen (bei Schrittweite 1) = Ende - Anfang + 1 */
        mw = summe / (ende-anf+1);

        /* Ausgabe der Ergebnisse */
        printf("\nSumme: %.0f",summe);
```

```
        printf("\nMittelwert: %.1f",mw);
}
```

Lösung zu Übung U0305:

```
/* Datei U0305.c */
#include <stdio.h>
void main(void)
{
        int x, anzahl, zahl, summe=0;

        /* Die gewünschte Anzahl der Zahlen soll eingegeben werden */
        printf("\nAnzahl eingeben: ");
        scanf("%i",&anzahl);

        /* Schleife von 1 bis "anzahl" in Schritten von 1 ergibt    */
        /* "anzahl" Durchläufe. Bei jedem Durchlauf wird eine Zahl */
        /* eingelesen                                               */
        for (x=1; x<=anzahl; x=x+1)
        {
             printf("%i. Zahl: ",x);
             scanf("%i",&zahl);
             summe = summe + zahl;
        }

        /* Ausgabe des Ergebnisses */
        printf("\nSumme: %i",summe);
}
```

Lösung zu Übung U0306:

```
/* Datei U0306.c */
#include <stdio.h>
void main(void)
{
        int x;

        /* Schleife von 5 bis ausschließlich 11, Schrittweite 1 */
        for (x=5; x<11; x=x+1)
             printf("Zahl: %i\n",x);
}
```

Lösung zu Übung U0307:

```
/* Datei U0307.c */
#include <stdio.h>
void main(void)
{
       int x;

       /* Schleife von 5 bis einschließlich 10, Schrittweite 1 */
       for (x=5; x<=10; x=x+1)
            printf("Zahl: %i\n",x);
}
```

Lösung zu Übung U0309:

```
/* Datei U0309.c */
#include <stdio.h>
void main(void)
{
       int x;

       /* Schleife von 24 bis ausschließlich 37, Schrittweite 3 */
       for (x=24; x<37; x=x+3)
            printf(" %i",x);
}
```

Lösung zu Übung U0310:

```
/* Datei U0310.c */
#include <stdio.h>
void main(void)
{
       int x;

       /* Abwärts-Schleife von 35 bis ausschließlich 29, Schrittweite -1 */
       for (x=35; x>29; x=x-1)
            printf(" %i",x);
}
```

Lösung zu Übung U0311:

```
/* Datei U0311.c */
#include <stdio.h>
void main(void)
```

```
{
        /* Die gewünschte Struktur ist zweidimensional, mit Zeilen und */
        /* Spalten. Daher werden zwei geschachtelte Schleifen benötigt */
        int x,y;

        /* 4 Durchläufe der äußeren Schleife, ergibt die Zeilen */
        for (x=0; x<4; x=x+1)
        {
             /* Ein Zeilensprung pro äußeren Schleifendurchlauf */
             printf("\n");

             /* 5 Durchläufe der inneren Schleife, alles in eine Zeile */
             for (y=24; y<37; y=y+3)
             {
                  /* Berechnung und Ausgabe der Zahl für jeden Wert     */
                  /* von x und jeden Wert von y, also 20mal             */
                  printf("%i ",x+y);
             }
        }
}
```

Lösung zu Übung U0312:

```
/* Datei U0312.c */
#include <stdio.h>
void main(void)
{
       /* Die Vorbesetzung von "fak" ist wegen der Rechnung in der    */
       /* Schleife erforderlich                                       */
       float x, zahl, fak=1;

       /* Eingabe der Zahl, deren Fakultät berechnet werden soll */
       printf("\nZahl eingeben: ");
       scanf("%f",&zahl);

       /* Fakultät von z.B. 6: 1*2*3*4*5*6, also alle Zahlen bis 6 */
       /* mit einer Schleife durchlaufen                            /
       for (x=1; x<=zahl; x=x+1)
       {
            /* Neuer Wert = Alter Wert * x             */
            /* Beim ersten Mal ist alter Wert gleich 1 */
            fak = fak * x;
       }

       /* Ausgabe des Ergebnisses */
       printf("\nFakultät von %.0f ist: %.0f", zahl, fak);
1}
```

20.4 Zu Kapitel 4

Lösung zu Übung U0402:

```
/* Datei U0402.c */
#include <stdio.h>
void main(void)
{
       int zahl;

       /* Die Variable "zahl" muss vorbesetzt werden, da ihr Wert in  */
       /* der while-Bedingung abgefragt wird                          */
       zahl=-1;

       /* Solange die aktuelle "zahl" ungleich 0 ist, wird die    */
       /* Schleife durchlaufen. Die Eingabe von 0 in der Schleife */
```

```
        /* führt zur letzten Ausgabe und zum Ende des Programmes  */
        while (zahl!=0)
        {
             printf("\nZahl eingeben ('0' für Ende): ");
             scanf("%i",&zahl);
             /* Quadrat der eingegebenen Zahl wird ausgegeben */
             printf("%3i hoch 2 = %4i\n", zahl, zahl*zahl);
        }
}
```

Lösung zu Übung U0403:

```
/* Datei U0403.c */
#include <stdio.h>
void main(void)
{
        double zahl;

        /* Die Zahl, die halbiert werden soll, wird eingelesen */
        printf("\nZahl eingeben: ");
        scanf("%lf",&zahl);

        /* Es wird nur halbiert, falls man noch oberhalb der Grenze liegt */
        while (zahl >= 0.001)
        {
             zahl = zahl / 2;
             /* Ausgabe der Zwischenergebnisse */
             printf("\nErgebnis: %12.7lf",zahl);
        }
}
```

Lösung zu Übung U0404:

```
/* Datei U0404.c */
#include <stdio.h>
void main(void)
{
        /* Es soll zusätzlich summiert werden */
        double zahl, summe=0;

        printf("\nZahl eingeben: ");
        scanf("%lf",&zahl);
```

```
        /* Schleifendurchlauf wie in 4.2 */
        while (zahl >= 0.001)
        {
             zahl = zahl / 2;
             /* Summierung */
             summe = summe + zahl;
             printf("\nErgebnis: %12.7lf",zahl);
        }

        /* Ausgabe des Ergebnisses */
        printf("\nSumme   : %12.7lf",summe);
}
```

Lösung zu Übung U0405:

```
/* Datei U0405.c */
#include <stdio.h>
void main(void)
{
        /* Es soll zusätzlich die Abbruchgrenze eingelesen werden */
```

```
        double zahl, summe=0, grenze;

        printf("\nZahl eingeben: ");
        scanf("%lf",&zahl);
        printf("\nGrenze eingeben: ");
        scanf("%lf",&grenze);

        /* Es wird nur halbiert, falls man noch oberhalb der Grenze liegt */
        while (zahl >= grenze)
        {
             zahl = zahl / 2;
             summe = summe + zahl;
             printf("\nErgebnis: %22.17lf",zahl);
        }

        /* Ausgabe des Eingabewertes und des Ergebnisses */
        printf("\nGrenze : %22.17lf",grenze);
        printf("\nSumme  : %22.17lf",summe);
}
```

Lösung zu Übung U0407:

```
/* Datei U0407.c */
#include <stdio.h>
void main(void)
{
        int zahl;

        /* Die Variable "zahl" muss jetzt nicht vorbesetzt werden, da  */
        /* sie auf jeden Fall in der Schleife, vor der while-Bedingung */
        /* besetzt wird. */
        do
        {
             printf("\nZahl eingeben ('0' für Ende): ");
             scanf("%i",&zahl);
             printf("%3i hoch 2 = %4i\n", zahl, zahl*zahl);
        }
        while (zahl!=0);
}
```

Lösung zu Übung U0408:

```
/* Datei U0408.c */
#include <stdio.h>
void main(void)
{
       double zahl;

       printf("\nZahl eingeben: ");
       scanf("%lf",&zahl);

       /* In diesem Fall wird mindestens einmal halbiert, da die      */
       /* Abfrage erst nach der Schleife kommt. "do-while" ist hier   */
       /* also schlechter geeignet als "while"                        */
       do
       {
            zahl = zahl / 2;
            printf("\nErgebnis: %12.7lf",zahl);
       }
       while (zahl >= 0.001);
}
```

Lösung zu Übung U0409:

```
/* Datei U0409.c */
#include <stdio.h>
void main(void)
{
       /* Startwert für while-Schleife */
       int i = 24;

       /* Abbruchwert für while-Schleife */
       while (i<37)
       {
            printf("\t%i",i);
            /* Schrittweite für while-Schleife */
            i=i+3;
       }
}
```

Lösung zu Übung U0410:

```
/* Datei U0410.c */
#include <stdio.h>
void main(void)
{
        float x, zahl, fak=1;

        /* Es wird nur weiter gearbeitet,     */
        /* wenn eine Zahl>0 eingegeben wurde */
        do
        {
             printf("\nZahl eingeben: ");
             scanf("%f",&zahl);
        }
        while (zahl <= 0);

        /* Startwert für do-while Schleife */
        x = 1;
        do
        {
```

```
            fak = fak * x;
            /* Schrittweite für do-while Schleife */
            x = x+1;
        }
        while (x <= zahl);     /* Endewert für do-while Schleife */

        /* Ausgabe des Ergebnisses */
        printf("\nFakultät von %.3f ist: %.3f", zahl, fak);
}
```

20.5 Zu Kapitel 5

Lösung zu Übung U0502:

```
/* Datei U0502.c */
#include <stdio.h>
void main(void)
{
        int j,x;

        /* Wiederholung, solange keine Null eingegeben wird */
        do
        {
            /* Einlesen der Zahl x */
            printf("Bitte Zahl eingeben: ");
            scanf("%i",&x);

            /* Ausgabe der x Sterne nur bei positiver Zahl x */
            if (x>0)
            {
                 for (j=1;j<=x;j++)
                      printf("*");
            }
            printf("\n");
        }
        while(x!=0);
}
```

Lösung zu Übung U0503:

```
/* Datei U0503.c */
#include <stdio.h>
void main(void)
```

```
{
        int x, ratezahl, eingabe;

        /* Einlesen der zu ratenden Zahl */
        printf("\t\tZ A H L E N R A T E N\n\n");
        printf("Welche Zahl soll erraten werden: ");
        scanf("%i",&ratezahl);

        /* 25 Zeilenvorschübe, damit die Zahl nicht mehr sichtbar ist */
        for (x=1;x<=25;x++)
             printf("\n");

        /* Wiederholung, solange die Zahl noch nicht gefunden wurde */
        do
        {
             /* Eingabe des Rateversuchs */
             printf("\nBitte Zahl eingeben: ");
             scanf("%i",&eingabe);

             /* Kommentar zum Rateversuch */
             if (eingabe > ratezahl)
                  printf("Zu groß\n");
             if (eingabe < ratezahl)
                  printf("Zu klein\n");
        }
        while(eingabe != ratezahl);

        /* Kommentar zum Ende des Programmes */
        printf("Zahl gefunden !\n");
}
```

Lösung zu Übung U0504:

```
/* Datei U0504.c */
#include <stdio.h>
void main(void)
{
        /* In "zg" und "zk" werden die Rateversuche gezählt */
        int x, ratezahl, eingabe, zg=0, zk=0;

        printf("\t\tZ A H L E N R A T E N\n\n");
        printf("Welche Zahl soll erraten werden: ");
```

```
scanf("%i",&ratezahl);
for  (x=1;x<=25;x++)
     printf("\n");

do
{
     printf("\nBitte  Zahl  eingeben:  ");
scanf("%i",&eingabe);

     if  (eingabe  >  ratezahl)
     {
          printf("Zu  groß\n");
          zg++;                              /*  Zähler  erhöhen  */
     }
     if  (eingabe  <  ratezahl)
     {
          printf("Zu  klein\n");
          zk++;                              /*  Zähler  erhöhen  */
     }
}
while(eingabe  !=  ratezahl);
```

```
        /* Kommentar zum Ende des Programmes */
        printf("Zahl gefunden !\n");

        /* Ausgabe der Zähler */
        printf("Insgesamt %i Versuche\n",zg+zk+1);
        printf("%2i mal zu groß\n",zg);
        printf("%2i mal zu klein\n",zk);
}
```

Lösung zu Übung U0506:

```
/* Datei U0506.c */
#include <stdio.h>
void main(void)
{
        int i;

        /* Insgesamt 8 Schleifendurchläufe */
        for (i=1;i<=8;i++)
        {
             /* Bei den ersten drei Durchläufen wird jedes */
             /* Mal "A" ausgegeben, danach jedes Mal "B"    */
             if (i<=3)
                  printf("A");
             else
                  printf("B");
        }
}
```

Lösung zu Übung U0508:

```
/* Datei U0508.c */
#include <stdio.h>
void main(void)
{
        int i;

        /* Insgesamt 9 Schleifendurchläufe */
        for (i=1;i<=9;i++)
        {
             /* Bei den ersten zwei Durchläufen wird jedes Mal "A"   */
             /* ausgegeben, danach etwas anderes                     */
```

```
            if (i<=2)
                 printf("A");
            else
            {
                 /* Bis zum 7. Durchlauf einschließlich wird jedes Mal  */
                 /* "B" ausgegeben, danach jedes Mal "C"                */
                 if (i<=7)
                      printf("B");
                 else
                      printf("C");
            }
       }
}
```

Lösung zu Übung U0509:

```
/* Datei U0509.c */
#include <stdio.h>
void main(void)
{
       float gehalt;

       /* Eingabedaten */
       printf("Bitte Gehalt eingeben: ");
       scanf("%f",&gehalt);

       /* Geschachtelte Fall-Unterscheidung, es wird insgesamt   */
       /* nur eine Ausgabe erzeugt                               */
       if (gehalt<=12000)
            printf("Steuerbetrag: %.2f",gehalt * 0.12);
       else

            /* alle Gehälter > 12000 */
            if (gehalt<=20000)
                 printf("Steuerbetrag: %.2f",gehalt * 0.15);
            else
                 /* alle Gehälter > 20000 */
                 if (gehalt<=30000)
                      printf("Steuerbetrag: %.2f",gehalt * 0.2);
                 else
                      printf("Steuerbetrag: %.2f",gehalt * 0.25);
}
```

Lösung zu Übung U0510:

```
/* Datei U0510.c */
#include <stdio.h>
void main(void)
{
       float gehalt, satz, stbetrag, vmgehalt;

       /* Überschrift der Tabelle */
       printf
       ("   Gehalt Steuersatz Steuerbetrag Gehalt abzgl. Steuer");

       /* Von 5000 bis 35000 bei Schrittweite 2000 */
       for (gehalt=5000; gehalt<36000; gehalt=gehalt+2000)
       {
            /* Geschachtelte Fall-Unterscheidung */
            if (gehalt<=12000)      satz=0.12;
            else if (gehalt<=20000) satz=0.15;
            else if (gehalt<=30000) satz=0.2;
            else                    satz=0.25;

            /* Berechnung und Ausgabe der Ergebnisse für jedes Gehalt */
```

```
            stbetrag = gehalt * satz;
            vmgehalt = gehalt - stbetrag;
            printf("\n%12.2f %12.0f %14.2f %22.2f",
                 gehalt, satz*100, stbetrag, vmgehalt);
       }
}
```

Lösung zu Übung U0511:

```
/* Datei U0511.c */
#include <stdio.h>
void main(void)
{
       int ug, og, i;

       /* Untergrenze muss > 0 sein */
       do
       {
            printf("Untergrenze eingeben: ");
            scanf("%i",&ug);
       }
       while (ug<=0);

       /* Obergrenze muss > Untergrenze sein */
       do
       {
            printf("Obergrenze eingeben: ");
            scanf("%i",&og);
       }
       while (og<=ug);

       /* Tabellenüberschrift */
       printf("\n Zahl  2     3     5     7  11\n");

       /* Schleife von Unter- bis Obergrenze bei Schrittweite 1 */
       for(i=ug;i<=og;i++)
       {
            printf("%5i",i);

            /* Falls die aktuelle Zahl glatt durch 2 teilbar ist, folgt */
            /* als Ausgabe: Drei Leerzeichen und ein x. Falls nicht,    */
            /* folgt die Ausgabe: Vier Leerzeichen                      */
```

```
            if (i%2==0)
                 printf("      x");
            else
                 printf("       ");

            if (i%3==0)
                 printf("      x");        /* Wie oben */
            else
                 printf("       ");

            if (i%5==0)
                 printf("      x");        /* Wie oben */
            else
                 printf("       ");

            if (i%7==0)
                 printf("      x");        /* Wie oben */
            else
                 printf("       ");

            if (i%11==0)
                 printf("      x");     /* Wie oben */
            else
                 printf("       ");

            /* Pro Schleifendurchlauf ein Zeilenvorschub */
            printf("\n");
        }
}
```

20.6 Zu Kapitel 6

Lösung zu Übung U0601:

- 1: wahr und falsch ergibt falsch
- 2: wahr oder falsch ergibt wahr
- 3: wahr oder falsch oder wahr ergibt wahr
- 4: wahr und falsch ergibt falsch, falsch oder wahr ergibt wahr
- 5: wahr und wahr ergibt wahr
- 6: wahr und nicht wahr ergibt falsch
- 7: wahr und wahr ergibt wahr, falsch und falsch ergibt falsch, wahr oder falsch ergibt wahr
- 8: wahr oder falsch ergibt wahr, wahr und wahr und falsch ergibt falsch

Lösung zu Übung U0602:

```
/* Datei U0602.c */
#include <stdio.h>
void main(void)
{
        double zahl;
        int counter=0;

        /* Zahl einlesen, die halbiert werden soll*/
        printf("\nZahl eingeben: ");
        scanf("%lf",&zahl);

        /* Halbieren und die Anzahl der Durchläufe mitzählen. */
        /* Halbiert wird, solange gilt: zahl > grenze und     */
        /* Zählerstand < 10, also maximal 10 Durchläufe       */
        do
        {
             zahl = zahl / 2;
```

```
            counter++;
            printf("\nNr. %3i,  Ergebnis: %18.7lf", counter, zahl);
        }
        while (zahl>=0.001 && counter<10);
}
```

Lösung zu Übung U0603:

```
/* Datei U0603.c */
#include <stdio.h>
void main(void)
{
        double zahl;
        int counter=0;
        printf("\nZahl eingeben: ");
        scanf("%lf",&zahl);

        /* Halbieren und die Anzahl der Durchläufe mitzählen. */
        /* Halbiert wird, solange gilt: zahl > grenze oder    */
        /* Zählerstand < 15, also minimal 15 Durchläufe       */
        do
        {
            zahl = zahl / 2;
            counter++;
            printf("\nNr. %3i,  Ergebnis: %18.7lf", counter, zahl);
        }
        while (zahl>=0.001 || counter<15);
}
```

Lösung zu Übung U0605:

```
/* Datei U0605.c */
#include <stdio.h>
void main(void)
{
        int x;

        do
        {
            printf("\nMenü Datei:\n\n");
            printf("1: Neu\n");
            printf("2: Öffnen\n");
            printf("3: Speichern\n");
            printf("4: Speichern unter\n");
```

```
            printf("5: Drucken\n");
            printf("6: Ende\n\n");

            /* Menüwunsch eingeben */
            printf("Bitte Menüpunkt auswählen\n");
            scanf("%i",&x);

            /* Auswahl und Ausgabe des betreffenden Menübefehls   */
            /* "break" sorgt dafür, dass nur eine Ausgabe erfolgt */
            switch(x)
            {
                 case 0: break;
                 case 1: printf("Neu            \n"); break;
                 case 2: printf("Öffnen         \n"); break;
                 case 3: printf("Speichern      \n"); break;
                 case 4: printf("Speichern unter\n"); break;
                 case 5: printf("Drucken        \n"); break;
                 case 6: printf("Ende           \n"); break;
                 default: printf("Nur von 0 bis 6 gültig!\n");
            }
       }
       while (x!=0); /* Solange keine 0 eingegeben wurde */
}
```

Lösung zu Übung U0606:

```
/* Datei U0606.c */
#include <stdio.h>
void main(void)
{
       double zahl; int i;

       /* Zahl eingeben, die halbiert werden soll */
       printf("\nZahl eingeben: ");
       scanf("%lf",&zahl);

       /* Maximal 10 Durchläufe */
       for (i=1;i<=10;i++)
       {
            zahl = zahl / 2;
            printf("\nNr. %3i,  Ergebnis: %18.7lf", i, zahl);
            /* Falls Ziel erreicht, Schleife vorzeitig verlassen */
```

```
            if (zahl<0.001) break;
        }
}
```

Lösung zu Übung U0608:

```
/* Datei U0608.c */
#include <stdio.h>
void main(void)
{
        int i;

        /* ASCII-Code 65 bis 90 entspricht A bis Z */
        /* Zahl mit Platzhalter %c ergibt das zugehörige Zeichen */
        for (i=65;i<=90;i++)
            printf("%c",i);

        /* ASCII-Code 97 bis 122 entspricht a bis z */
        for (i=97;i<=122;i++)
            printf("%c",i);
}
```

Lösung zu Übung U0609:

```
/* Datei U0609.c */
#include <stdio.h>
void main(void)
{
       int x;
       for (x=0;x<=255;x++)
       {
            if (!(x>=7 && x<=10) && x!=13 && x!=26 && x!=27)
                 printf("%4i %c",x,x);
            else
                 printf("      ");

            if(x%8==7)
                 printf("\n");
       }
       printf("\n");
}
```

20.7 Zu Kapitel 7

Lösung zu Übung U0702:

```
/* Datei U0702.c */
#include <stdio.h>

void box(void)
{
       /* Pro Zeile drei Sonderzeichen (Rahmen) */
       printf("\n%c%c%c",201,205,187);
       printf("\n%c%c%c",186, 32,186);
       printf("\n%c%c%c",200,205,188);
}

void main(void)
{
       int x;
       /* Dreimal die Funktion box() aufrufen */
       for (x=1;x<=3;x++)
            box();
}
```

Lösung zu Übung U0704:

```
/* Datei U0704.c */
#include <stdio.h>

/* Funktion zur Multiplikation von zwei übergebenen */
/* double-Zahlen und zur Ausgabe des Ergebnisses    */
void mult(double x, double y)
{
       double z;
       z = x*y;
       printf("\n%lf * %lf = %lf",x,y,z);
}

/* Haupt-Programm */
void main(void)
{
       double a,b;

       /* Eingabe von zwei double-Zahlen */
       printf("1. Zahl: ");
       scanf("%lf",&a);
       printf("2. Zahl: ");
       scanf("%lf",&b);

       /* Aufruf der Funktion */
       mult(a,b);
}
```

Lösung zu Übung U0705:

```
/* Datei U0705.c */
#include <stdio.h>

/* Funktion zur Berechnung des Mittelwertes von drei */
/* float-Zahlen und zur Ausgabe des Ergebnisses      */
void mw(float x, float y, float z)
{
       float a;
       a = (x+y+z)/3;
       printf("\nMittelwert: %f",a);
}
```

```
/* Haupt-Programm */
void main(void)
{
        float a,b,c;

        /* Eingabe von drei float-Zahlen */
        printf("1. Zahl: ");
        scanf("%f",&a);
        printf("2. Zahl: ");
        scanf("%f",&b);
        printf("3. Zahl: ");
        scanf("%f",&c);

        /* Aufruf der Funktion */
        mw(a,b,c);
}
```

Lösung zu Übung U0707:

```
/* Datei U0707.c */
#include <stdio.h>

/* Funktion zur Berechnung des Mittelwertes von drei */
/* float-Zahlen und zur Rückgabe des Ergebnisses an  */
/* das aufrufende Programm                           */
float mw(float x, float y, float z)
{
       float a;
       a = (x+y+z)/3;
       return(a);
}

/* Haupt-Programm */
void main(void)
{
       float a,b,c,z;

       printf("1. Zahl: ");
       scanf("%f",&a);
       printf("2. Zahl: ");
       scanf("%f",&b);
       printf("3. Zahl: ");
       scanf("%f",&c);

       /* Aufruf der Funktion und Zuweisung des */
       /* Ergebnisses zur Variablen z           */
       z=mw(a,b,c);

       /* Ausgabe des Ergebnisses */
       printf("\nMittelwert: %f",z);
}
```

Lösung zu Übung U0709:

```
/* Datei U0709.c */
#include <stdio.h>

/* Prototyp der Funktion mw(), dadurch ist */
/* der Name "mw" im Hauptprogramm bekannt  */
float mw(float, float, float);
```

```
/* Haupt-Programm, Inhalt wie in der vorherigen Übung. */
void main(void)
{
       float a,b,c,z;
       printf("1. Zahl: ");
       scanf("%f",&a);
       printf("2. Zahl: ");
       scanf("%f",&b);
       printf("3. Zahl: ");
       scanf("%f",&c);
       z=mw(a,b,c);
       printf("\nMittelwert: %f",z);
}

/* Funktion mw(), Inhalt wie in der vorherigen Übung. */
/* Wegen des Prototypes kann die Funktion unterhalb   */
/* des Hauptprogrammes geschrieben werden             */
float mw(float x, float y, float z)
{
       float a;
       a = (x+y+z)/3;
       return(a);
}
```

Lösung zu Übung U0710:

```
/* Datei U0710.c */
#include <stdio.h>
int vokal(char);

void main(void)
{
       char x;
       for(x='a'; x<='r'; x++)
            printf("%c %i\n",x,vokal(x));
       printf("\n");
}

int vokal(char z)
{
       switch(z)
       {
```

```
            case 'a':
            case 'e':
            case 'i':
            case 'o':
            case 'u':
            case 'A':
            case 'E':
            case 'I':
            case 'O':
            case 'U': return(1);
            default:  return(0);
        }
}
```

Lösung zu Übung U0711:

```
/* Datei U0711.c */
#include <stdio.h>

int vorzeichen(double);
```

```
void main(void)
{
       double a;
       for(a=5.5; a>-7.2; a=a-0.78)
            printf("%3i %10.3lf\n",vorzeichen(a),a);
}

int vorzeichen(double z)
{
       if (z>0)         return(1);
       else if (z<0)   return(-1);
       else             return(0);
}
```

Lösung zu Übung U0712:

```
/* Datei U0712.c */
#include <stdio.h>

void box(int hoehe, int breite)
{
       int i,j;

       printf("\n%c",201);
       for (j=1;j<=breite-2;j++)
            printf("%c",205);
       printf("%c",187);

       for (i=1;i<=hoehe-2;i++)
       {
            printf("\n%c",186);
            for (j=1;j<=breite-2;j++)
                 printf("%c",32);
            printf("%c",186);
       }

       printf("\n%c",200);
       for (j=1;j<=breite-2;j++)
            printf("%c",205);
       printf("%c",188);
}
```

```
void main(void)
{
        int h,b;
        int antwort;

        do
        {
             printf("\nHoehe eingeben: ");
             scanf("%i",&h);
             printf("Breite eingeben: ");
             scanf("%i",&b);

             if (h>=2 && h<=20 && b>=2 && b<=79)
                  box(h,b);
             else
                  printf("Hoehe von 2 bis 20, Breite von 2 bis 79 !");

             do
             {
                  printf("\n\nWeiter (1=Weiter / 2=Nicht weiter) ? ");
                  scanf("%i",&antwort);
             }
             while (antwort!=1 && antwort!=2);
        }
        while (antwort==1);
}
```

20.8 Zu Kapitel 8

Lösung zu Übung U0802:

```
/* Datei U0802.c */
#include <stdio.h>
void main(void)
{
        /* Feld x beinhaltet 5 double-Zahlen, Index von 0 bis 4 */
        double x[5];
        double summe=0, mw;
        int i;

        /* Schleife zum Einlesen der 5 double-Zahlen, Index i von 0 bis 4 */
        for (i=0;i<5;i++)
        {
```

```
            printf("Wert %i eingeben: ",i);
            scanf("%lf",&x[i]);       /* Einlesen der i-ten double-Zahl */
            summe += x[i];          /* Summieren der i-ten double-Zahl */
        }

        /* Berechnung und Ausgabe der Ergebnisse */
        mw = summe / 5;
        printf("Summe: %lf",summe);
        printf("\nMittelwert: %lf",mw);
}
```

Lösung zu Übung U0803:

```
/* Datei U0803.c */
#include <stdio.h>
void main(void)
{
        double x[5], minwert;
        int i;

        /* Alle 5 Feldelemente einlesen */
        for (i=0;i<5;i++)
        {
            printf("Wert %i eingeben: ",i);
            scanf("%lf",&x[i]);
        }

        /* Erste Annahme: Minimum ist Feldelement 0 */
        minwert = x[0];

        /* Untersuchung der restlichen Feldelemente */
        for (i=1;i<5;i++)
        {
            /* Falls eines der restlichen Feldelemente kleiner ist, */
            /* wird dieses zum neuen Minimum                        */
            if (x[i] < minwert) minwert = x[i];
        }

        /* Ausgabe des Minimums */
        printf("Minimum: %lf",minwert);
}
```

Lösung zu Übung U0804:

```
/* Datei U0804.c */
#include <stdio.h>
void main(void)
{
        double x[5], y[5];
        int i;

        /* Alle 5 Feldelemente einlesen */
        for (i=0;i<5;i++)
        {
             printf("x[%i] eingeben: ",i);
             scanf("%lf",&x[i]);
        }

        /* Zweites Feld in umgekehrter Reihenfolge mit den Elementen */
        /* des ersten Feldes belegen: 0->4, 1->3, 2->2 usw.          */
        for (i=0;i<5;i++)
             y[4-i] = x[i];

        /* Beide Felder nebeneinander ausgeben */
```

```
        for (i=0;i<5;i++)
             printf("\nx[%i]: %6.2lf y[%i]: %6.2lf",i,x[i],i,y[i]);
}
```

Lösung zu Übung U0805:

```
/* Datei U0805.c */
#include <stdio.h>
void main(void)
{
        double x[5];
        int i, minpos;

        /* Alle 5 Feldelemente einlesen */
        for (i=0;i<5;i++)
        {
             printf("Wert %i eingeben: ",i);
             scanf("%lf",&x[i]);
        }

        /* Erste Annahme: Minimum ist an Feldposition 0 */
        minpos = 0;

        /* Untersuchung der restlichen Feldelemente */
        for (i=1;i<5;i++)
        {
             /* Falls eines der restlichen Feldelemente kleiner ist, */
             /* wird dessen Position zur neuen Minimum-Position     */
             if (x[i] < x[minpos])
                  minpos = i;
        }

        /* Ausgabe des Ergebnisses */
        printf("Minimum bei Position: %i",minpos);
}
```

Lösung zu Übung U0806:

```
/* Datei U0806.c */
#include <stdio.h>
void main(void)
{
        double x[5], y[5], summe = 0;
```

```
        int i;

        /* Alle 5 Feldelemente einlesen */
        for (i=0;i<5;i++)
        {
             printf("Wert %i eingeben: ",i);
             scanf("%lf",&x[i]);
        }

        /* Berechnung der Mittelwerte */
        for (i=0;i<5;i++)
        {
             /* Zwischensumme berechnen */
             summe += x[i];

             /* Zwischen-Mittelwert berechnen und dem Element */
             /* des zweiten Feldes zuweisen                   */
             y[i] = summe / (i+1);
        }

        /* Ausgabe der beiden Felder */
        for (i=0;i<5;i++)
             printf("\nx[%i]: %6.2lf y[%i]: %6.2lf",i,x[i],i,y[i]);
}
```

Lösung zu Übung U0807:

```
/* Datei U0807.c */
#include <stdio.h>

/* Funktion bekommt zwei double-Zahlen und gibt eine zurück */
double vglmin(double,double);

/* Haupt-Programm */
void main(void)
{
        double x[5], minwert;
        int i;

        for (i=0;i<5;i++)
        {
             printf("Wert %i eingeben: ",i);
```

```
            scanf("%lf",&x[i]);
        }

        /* Annahme: Minimum ist erstes Feldelement */
        minwert = x[0];

        /* Für die restlichen Feldelemente wird die Funktion vglmin()   */
        /* aufgerufen. Diese liefert das neue Minimum an minwert zurück */
        for (i=1;i<5;i++)
            minwert = vglmin(minwert,x[i]);

        /* Ausgabe des Ergebnisses */
        printf("Minimum: %lf",minwert);
}

/* Funktion vglmin() bekommt zwei double-Zahlen */
/* und liefert die kleinere von beiden zurück.  */
double vglmin(double a,double b)
{
        if(a<b) return (a);
        else  return(b);
}
```

Lösung zu Übung U0808:

```
/* Datei U0808.c */
#include <stdio.h>
void main(void)
{
        int x[5], i, maxwert, anzpos;

        /* Feld zur Speicherung von maximal 5 Positionen */
        /* Feldelement 0 wird nicht benutzt              */
        int maxpos[6];

        for (i=0;i<5;i++)
        {
            printf("Wert %i eingeben: ",i);
            scanf("%i",&x[i]);
        }

        /* Annahme: Das Maximum ist das erste Feldelement. Die Anzahl der  */
        /* Maxima ist 1. Die Feld-Position des Maximums (0) wird als       */
```

```
/* erste Position eines Maximums gespeichert.                    */
maxwert          = x[0];
anzpos           = 1;
maxpos[anzpos]   = 0;

/* Untersuchung der restlichen Feldelemente */
for (i=1;i<5;i++)
{
     /* Falls ein neues Maximum an Position i gefunden wurde */
     if (x[i] > maxwert)
     {
          maxwert = x[i];
          anzpos = 1;
          maxpos[anzpos] = i;
     }

     /* Falls das bisherige Maximum noch einmal gefunden wurde */
     else if (x[i] == maxwert)
     {
          anzpos++;           /* Anzahl Maxima erhöhen */
          maxpos[anzpos] = i;/* Position speichern    */
```

```
            }
        }

        /* Ausgabe aller Maxima */
        for (i=1;i<=anzpos;i++)
            printf("\n%i. Maximum bei Position: %i",i,maxpos[i]);
}
```

20.9 Zu Kapitel 9

Lösung zu Übung U0905:

```
/* Datei U0905.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char txa[20], txb[10], txc[10];

        strcpy(txa,"Struk");
        strcpy(txb,"to");
        strcpy(txc,"gramm");

        printf("\n%s",txa);
        printf("\n%i",strlen(txa));
        printf("\n%s",txb);
        printf("\n%i",strlen(txb));
        printf("\n%s",txc);
        printf("\n%i",strlen(txc));

        strcat(txa,txb);
        strcat(txa,txc);
        printf("\n%s",txa);
        printf("\n%i",strlen(txa));
        printf("\n");
}
```

Lösung zu Übung U0906:

```
/* Datei U0906.c */
#include <stdio.h>
#include <string.h>
```

```
void main(void)
{
        char txa[30], txb[10];
        int i;

        strcpy(txa,"Programmiertechnik");

        txa[15] = '\0';
        for (i=0;i<30;i++)
             printf("%c",txa[i]);
        printf("\n%s",txa);
        printf("\n%i\n",strlen(txa));

        txa[12] = '\0';
        for (i=0;i<30;i++)
             printf("%c",txa[i]);
        printf("\n%s",txa);
        printf("\n%i\n",strlen(txa));

        txa[5] = '\0';
        for (i=0;i<30;i++)
             printf("%c",txa[i]);
        printf("\n%s",txa);
        printf("\n%i\n",strlen(txa));
}
```

Lösung zu Übung U0907:

```
/* Datei U0907.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char txa[80];
        int i, laenge;
        float zaehl=0;
        gets(txa);
        laenge = strlen(txa);

        for (i=0; i<laenge; i++)
             if (txa[i]=='e') zaehl++;
```

```
        printf("Das Zeichen e kommt %.0f mal vor", zaehl);
        printf("\nDas sind %.2f%% aller Zeichen",zaehl/laenge*100);
        printf("\nDer Text hat insgesamt %i Zeichen\n",laenge);
}
```

Lösung zu Übung U0908:

```
/* Datei U0908.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char txa[80];
        int i, laenge;
        gets(txa);
        laenge = strlen(txa);

        for (i=0; i<laenge; i++)
             if (txa[i] != ' ')
                  printf("%c",txa[i]);

        printf("\n");
}
```

Lösung zu Übung U0909:

```
/* Datei U0909.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
       char geheim[20],versuch[20];
       int i, laenge, korrekt;

       do
       {
            printf("Wort eingeben (genau drei kleine Buchstaben): ");
            gets(geheim);
            laenge = strlen(geheim);

            korrekt = 1;
            if (laenge != 3) korrekt = 0;
            for(i=0; i<3; i++)
            {
                 if (geheim[i] < 'a' || geheim[i] > 'z')
                 {
                      korrekt = 0;
                      break;
                 }
            }
            if (korrekt == 0)
                 printf("Genau drei kleine Buchstaben !\n");
       }
       while(korrekt == 0);

       for(i=1; i<=30; i++)
            printf("\n");

       do
       {
            printf("\nVersuch eingeben: ");
            gets(versuch);
            if (strcmp(versuch,geheim) < 0)
                 printf("Versuch < Passwort");
            else
```

```
                if (strcmp(versuch,geheim) > 0)
                     printf("Versuch > Passwort");
        }
        while(strcmp(versuch,geheim)!=0);

        printf("\nGefunden\n");
}
```

Lösung zu Übung U0910:

```
/* Datei U0910.c */
#include <stdio.h>
#include <string.h>

char umw(char);

void main(void)
{
        int i, laenge;
        char tx[80];

        printf("Text eingeben: ");
        gets(tx);
        laenge = strlen(tx);

        for(i=0; i<laenge; i++)
             printf("%c",umw(tx[i]));
        printf("\n");
}
char umw(char ze)
{
        char za;
        if (ze>='a' && ze<='z') za = ze - 32;
        else                    za = ze;
        return(za);
}
```

20.10 Zu Kapitel 10

Lösung zu Übung U1001:

```
/* Datei U1001.c */
#include <stdio.h>
void main(void)
```

```
{
        int i;
        double c,f,anf,ende,diff;

        /* Eingabe der Grenzen */
        printf("Geben Sie die erste Grenze ein: ");
        scanf("%lf",&anf);
        printf("\nGeben Sie die zweite Grenze ein: ");
        scanf("%lf",&ende);

        /* Berechnen des Abstandes, 15 Zahlen beinhalten 14 Abstände */
        diff = (ende - anf) / 14;

        /* 15 Schleifen-Durchläufe */
        c = anf;                    /* Startwert für Celsius-Wert */
        for(i=1;i<=15;i++)
        {
             /* Berechnung und Ausgabe des Fahrenheit-Wertes */
```

```
            f = c * 9 / 5 + 32;
            printf("\n %3i %10.4lf %10.4lf",i,c,f);

            /* Erhöhung des Celsius-Wertes um die Differenz */
            c = c + diff;
        }
}
```

Lösung zu Übung U1002:

```
/* Datei U1002.c */
#include <stdio.h>
void main(void)
{
        int i, k, diff;
        int t[2], m[2];             /* Tag und Monat der beiden Daten */
        int sum[2]={0,0};           /* Zwei Hilfs-Summen */

        /* Tage der jeweiligen Monate */
        int anzt[13] = {0,31,28,31,30,31,30,31,31,30,31,30,31};

        /* Eingabedaten, es werden nur sinnvolle Daten angenommen */
        for (i=0;i<2;i++)
        {
            do
            {
                printf("\nMonat des %i. Datums: ",i+1);
                scanf("%i",&m[i]);
            }
            while (m[i] < 1 || m[i] > 12);

            do
            {
                printf("\nTag des %i. Datums: ",i+1);
                scanf("%i",&t[i]);
            }
            while (t[i] < 1 || t[i] > anzt[m[i]]);
        }

        /* Für jedes Datum einzeln: Tag des Jahres berechnen */
        for (i=0;i<2;i++)
        {
```

```
            /* Alle vollen Monate addieren */
            for (k=1; k<m[i]; k++)
                sum[i] = sum[i] + anzt[k];

            /* Tage des Datums addieren */
            sum[i] = sum[i] + t[i];
        }

        /* Differenz der Tage des Jahres berechnen */
        diff = sum[1] - sum[0];

        /* Betrag bilden */
        if (diff < 0)
            diff = diff * (-1);

        /* Ausgabe */
        printf("\nDifferenz zwischen %i.%i. und %i.%i.: %i Tage",
            t[0],m[0],t[1],m[1],diff);
}
```

Lösung zu Übung U1003:

```
/* Datei U1003.c */
#include <stdio.h>
void main(void)
{
        int i,z;

        /* Oberer Rahmen */
        printf("\n%c%c%c%c%c%c",218,196,196,196,196,194);
        for (i=0;i<31;i++)
            printf("%c",196);
        printf("%c",191);

        /* Überschrift */
        printf("\n%c      %c",179,179);
        for (i=0;i<10;i++)
            printf("%3i",i+1);
        printf(" %c",179);

        /* Zwischenrahmen */
        printf("\n%c%c%c%c%c%c",195,196,196,196,196,197);
```

```
        for (i=0;i<31;i++)
             printf("%c",196);
        printf("%c",180);

        /* Eigentliche Tabelle */
        for (z=1;z<=5;z++)
        {
             printf("\n%c%3i %c",179,z,179);
             for (i=0;i<10;i++)
                  printf("%3i",(i+1)*z);
             printf(" %c",179);
        }

        /* Unterer Rahmen */
        printf("\n%c%c%c%c%c%c",192,196,196,196,196,193);
        for (i=0;i<31;i++)
             printf("%c",196);
        printf("%c",217);
}
```

Lösung zu Übung U1004:

- Äußere Schleife: 5 Durchläufe mit x von -3 bis +1 in Schritten von +1, pro Durchlauf ein Zeilensprung, also pro x eine Zeile Ausgabe
- Innere Schleife: je 5 Durchläufe mit y von +7 bis +3 in Schritten von -1, pro Durchlauf eine Ausgabe des berechneten Ausdrucks, also pro Zeile 5 Zahlen Ausgabe
- es ergeben sich 5 mal 5 Zahlen
- diese erhöhen sich innerhalb einer Zeile um 1, weil y im berechneten Ausdruck mit Faktor 1 eingeht
- sie erhöhen sich innerhalb einer Spalte um 2, weil x im berechneten Ausdruck mit Faktor 2 eingeht

1 0 -1 -2 -3

3 2 1 0 -1

5 4 3 2 1

7 6 5 4 3

9 8 7 6 5

Lösung zu Übung U1005:

- int prod=1; (Initialisierung für das Produkt wurde vergessen)
- scanf("%i",&zahl); (scanf muss mit & geschrieben werden)
- printf("Produkt: %i",prod); (Ende der Zeichenkette " wurde vergessen)

Lösung zu Übung U1006:

- i durchläuft die Werte 5 7 9 11 13 15 17 19 21 23 (also 10 Durchläufe)
- für alle i kleiner 18 trifft äußere if-Bedingung zu
- falls diese zutraf: für alle i kleiner 12 trifft innere if-Bedingung zu
- je nach Bedingung wird entweder ein X oder ein Y oder ein Z ausgegeben, in jedem Fall genau ein Zeichen

XXXXYYYZZZ

Lösung zu Übung U1007:

- double kw(double) (die Funktion muss vom Typ double sein, dies wurde beim Prototyp falsch angegeben)
- scanf("%lf",&b) (es wird in b statt in a eingelesen in der Schleife)
- double y (es fehlt die Deklaration des y in der Funktion)

Lösung zu Übung U1008:

```
/* Datei U1008.c */
#include <stdio.h>
void main(void)
{
       int anz, i, getauscht;
       double zfeld[10], temp;

       /* Anzahl einlesen, nur sinnvolle Daten */
       do
       {
            printf("\nWie viele Zahlen: ");
            scanf("%i",&anz);
       }
       while (anz<1 || anz>10);

       /* Zahlen einlesen */
       for(i=0;i<anz;i++)
       {
            printf("Zahl %i: ",i);
            scanf("%lf",&zfeld[i]);
       }

       /* Wiederholen, falls getauscht wurde */
       do
       {
            getauscht = 0;           /* Bisher wurde nicht getauscht     */

            /* Alle Zahlen von der ersten bis zur vorletzten durchgehen */
            for(i=0;i<anz-1;i++)
            {
                 /* Ist die aktuelle Zahl größer als ihr Nachfolger ?/
                 if(zfeld[i] > zfeld[i+1])
                 {
                      /* Tauschen */
                      temp = zfeld[i];
                      zfeld[i] = zfeld[i+1];
                      zfeld[i+1] = temp;
                      getauscht = 1;      /* Es wurde getauscht */
                 }
            }
```

```
        }
        /* Falls mindestens einmal getauscht wurde, wiederholen */
        while(getauscht);

        /* Sortierte Zahlen ausgeben */
        for(i=0;i<anz;i++)
             printf("\nZahl %i: %10.3lf",i,zfeld[i]);
}
```

20.11 Zu Kapitel 11

Lösung zu Übung U1102:

```
/* Datei U1102.c */
#include <stdio.h>
void main(void)
{
        int i, j, a[5][8];

        /* Geschachtelte Schleife, damit sowohl alle Zeilen als auch */
        /* alle Spalten durchlaufen werden. Der Wert jedes Elementes */
        /* ist nur von der Zeilennummer i abhängig, von Zeile zu     */
        /* Zeile um 4 erhöht, startet bei 6                          */
```

```
        for (i=0;i<5;i++)                   /* alle Zeilen  */
             for (j=0;j<8;j++)              /* alle Spalten */
                  a[i][j] = i*4 + 6;

        printf("Das Feld hat folgenden Inhalt:\n");
        /* Ausgabe: alle Zeilen, alle Spalten */
        for (i=0;i<5;i++)
        {
             for (j=0;j<8;j++)
                  printf("%3i ", a[i][j]);
             printf("\n");            /* pro Zeile ein Zeilenvorschub */
        }
}
```

Lösung zu Übung U1103:

```
/* Datei U1103.c */
#include <stdio.h>
void main(void)
{
        int eme[10][10], i, j;

        /* 10 Zeilen, 10 Spalten                                    */
        /* Wert jedes Elementes: Zeilennummer i mal Spaltennummer j */
        /* Jeweils Addition von 1, da i und j bei 0 beginnen.           */
        for(i=0;i<10;i++)
             for(j=0;j<10;j++)
                  eme[i][j] = (i+1)*(j+1);

        /* Ausgabe */
        for(i=0;i<10;i++)
        {
             for(j=0;j<10;j++)
                  printf("%3i ", eme[i][j]);
             printf("\n");     /* pro Zeile ein Zeilenvorschub */
        }
}
```

Lösung zu Übung U1104:

```
/* Datei U1104.c */
#include <stdio.h>
void main(void)
```

```
{
        int eme[10][10], temp;
        int i, j;

        /* Berechnung der 10 mal 10 Zahlen */
        for(i=0;i<10;i++)
             for(j=0;j<10;j++)
                  eme[i][j]=(i+1)*(j+1);

        /* Eine Schleife mit 10 Durchläufen zum Austausch der zehn */
        /* Zahlen der beiden letzten Spalten                       */
        for(i=0;i<10;i++)
        {
             temp = eme[i][9];
             eme[i][9] = eme[i][8];
             eme[i][8] = temp;
        }

        /* Ausgabe */
        for(i=0;i<10;i++)
        {
             for(j=0;j<10;j++)
                  printf("%3i ", eme[i][j]);
             printf("\n");
        }
}
```

Lösung zu Übung U1105:

```
/* Datei U1105.c */
#include <stdio.h>
void main(void)
{
        int eme[10][10], temp;
        int i, j, z1, z2;

        /* Berechnung der 10 mal 10 Zahlen */
        for(i=0;i<10;i++)
             for(j=0;j<10;j++)
                  eme[i][j]=(i+1)*(j+1);

        /* Eingabe der Indizes der beiden zu vertauschenden Zeilen. */
```

```
/* Es werden nur sinnvolle Eingaben angenommen                */
do
{
     printf("Geben Sie die Nummern ein (0 bis 9): ");
     printf("\n1. Zeile: ");
     scanf("%i",&z1);
     printf("2. Zeile: ");
     scanf("%i",&z2);
}
while(z1==z2 || z1<0 || z1>9 || z2<0 || z2>9);

/* Eine Schleife mit 10 Durchläufen zum Austausch der zehn */
/* Zahlen der beiden auszutauschenden Zeilen                  */
for(j=0;j<10;j++)
{
     temp = eme[z1][j];
     eme[z1][j] = eme[z2][j];
     eme[z2][j] = temp;
}

/* Ausgabe */
```

```
        for(i=0;i<10;i++)
        {
             for(j=0;j<10;j++)
                  printf("%3i ", eme[i][j]);
             printf("\n");
        }
}
```

Lösung zu Übung U1106:

```
/* Datei U1106.c */
#include <stdio.h>
void main(void)
{
        /* Eindimensionale Felder                          */
        /* Größenangabe darf fehlen bei Initialisierung */
        int a[3] = {3,4,5};
        int b[5] = {2,4,6};
        double c[5] = {4.2,5.3,6.4,7.5};
        int d[] = {-1,5,9,22,4};

        /* Zweidimensionale Felder,                              */
        /* eine Größenangabe darf fehlen bei Initialisierung */
        int e[2][3] = {{3,4,5},{9,8,7}};
        int f[][3] = {{3,4},{9,8,7},{0},{4,8}};

        int i,j;

        for (i=0;i<3;i++)
             printf("%3i",a[i]);
        printf("\n");

        for (i=0;i<5;i++)
             printf("%3i",b[i]);
        printf("\n");

        for (i=0;i<5;i++)
             printf("%6.2lf",c[i]);
        printf("\n");

        for (i=0;i<5;i++)
             printf("%3i",d[i]);
```

```
        printf("\n");

        for (i=0;i<2;i++)
        {
             for (j=0;j<3;j++)
                  printf("%3i",e[i][j]);
             printf("\n");
        }

        for (i=0;i<4;i++)
        {
             for (j=0;j<3;j++)
                  printf("%3i",f[i][j]);
             printf("\n");
        }
}
```

20.12 Zu Kapitel 12

Lösung zu Übung U1201:

```
/* Datei U1202.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char tx[4][20];
        int i;

        strcpy(tx[0],"Struk");
        strcpy(tx[1],"to");
        strcpy(tx[2],"gramm");

        strcpy(tx[3],tx[0]);
        strcat(tx[3],tx[1]);
        strcat(tx[3],tx[2]);

        for (i=0;i<4;i++)
             printf("\n%s, Laenge: %i",tx[i], strlen(tx[i]));

        printf("\n");
}
```

Lösung zu Übung U1203:

```
/* Datei U1203.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
       char ein[21], aus[20][21];
       int laenge, i,j;

       /* Eingabe der Zeichenkette, Länge feststellen */
       gets(ein);
       laenge = strlen(ein);

       /* Erste Hälfte der Ausgabe */
       for(i=0;i<laenge;i++)
       {
            strcpy(aus[i],ein);  /* Kopie der ganzen Zeichenkette */

            /* Einsetzen des ZK-Ende an der geeigneten Stelle, */
            /* dadurch wird die ZK verkürzt                    */
            aus[i][i+1] = ,\0';

            /* Ausgabe der verkürzten ZK */
            printf("\n%s",aus[i]);
       }

       /* Zweite Hälfte der Ausgabe, in umgekehrter Reihenfolge */
       for(i=laenge-2;i>=0;i--)
            printf("\n%s",aus[i]);
}
```

Lösung zu Übung U1204:

```
/* Datei U1204.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
       char ein[3][76];
       int i, j;
```

```
int count[3][5];              /* jeden Vokal pro Satz */
int sumvok;                   /* Zur Berechnung der Anteile */
int sumcount[5];              /* jeden Vokal über alle Sätze */

/* alle Zähler auf Null stellen */
for(i=0;i<3;i++)
     for(j=0;j<5;j++)
     {
          count[i][j]=0;
          sumcount[j] = 0;
     }

/* 3 Sätze einlesen */
for(i=0;i<3;i++)
{
     printf("Satz %i: ",i);
     gets(ein[i]);
}

/* Sätze untersuchen, Vokale zählen */
for(i=0;i<3;i++)
```

```
        for(j=0;j<strlen(ein[i]);j++)
        {
            switch(ein[i][j])
            {
                case 'a':
                case 'A': count[i][0]++; break;
                case 'e':
                case 'E': count[i][1]++; break;
                case 'i':
                case 'I': count[i][2]++; break;
                case 'o':
                case 'O': count[i][3]++; break;
                case 'u':
                case 'U': count[i][4]++; break;
            }
        }

    /* Tabellenüberschrift */
    printf("\n a     e     i     o     u\n\n");

    /* Anzahlen und Anteile ausgeben */
    for(i=0;i<3;i++)
    {
        sumvok = 0;         /* Zur Berechnung der Anteile */
        for(j=0;j<5;j++)
        {
            /* Einzelzähler ausgeben */
            printf("%4i",count[i][j]);

            /* Summierung für Anteile */
            sumvok = sumvok + count[i][j];

            /* Summierung für Gesamt-Zähler */
            sumcount[j] = sumcount[j] + count[i][j];
        }

        /* Berechnung und Ausgabe der Anteile */
        printf("    Anteil der Vokale: %i Prozent\n",
            sumvok * 100 / strlen(ein[i]));
    }

    /* Gesamtzähler ausgeben */
```

```
        printf("\n");
        for(j=0;j<5;j++)
             printf("%4i",sumcount[j]);
}
```

Lösung zu Übung U1205:

```
/* Datei U1205.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char name[5][21], temp[21];

        i, getauscht;

        /* 5 Namen einlesen */
        for(i=0;i<5;i++)
        {
             printf("Name %i: ",i);
             gets(name[i]);
        }

        /* Solange wiederholen, bis nicht mehr getauscht wurde */
        do
        {
             /* Bisher wurde nicht getauscht */
             getauscht = 0;

             /* Erste bis vorletzte Zeichenkette untersuchen */
             for(i=0;i<4;i++)
             {
                  /* Steht diese ZK alphabetisch hinter ihrem Nachfolger? */
                  if(strcmp(name[i],name[i+1])>0)
                  {
                       /* Tauschen */
                       strcpy(temp,name[i]);
                       strcpy(name[i],name[i+1]);
                       strcpy(name[i+1],temp);
                       getauscht = 1;  /* Es wurde getauscht */
                  }
```

```
            }
        }
        while(getauscht);

        /* Ausgabe der alphabetisch sortierten Zeichenketten */
        for(i=0;i<5;i++)
            printf("\nName %i: %s",i,name[i]);
}
```

Lösung zu Übung U1206:

```
/* Datei U1206.c */
#include <stdio.h>
#include <string.h>

void main(void)
{
        char name[4][6][20], temp[20];
        int i, j, k, getauscht, leerstellen;
```

```
/* Namen zuordnen */
strcpy(name[0][0],"Benders");
strcpy(name[0][1],"Bohnen");
strcpy(name[0][2],"Wachenhusen");
strcpy(name[0][3],"Pesce");
strcpy(name[1][0],"Kranz");
strcpy(name[1][1],"Morcinek");
strcpy(name[1][2],"Esser");
strcpy(name[1][3],"Rosemann");

/* Tauschvorgang, beide Reihen getrennt tauschen */
for(i=0;i<2;i++)
     do
     {
          /* Bisher noch nicht getauscht */
          getauscht = 0;

          /* Erste bis vorletzte ZK pro Reihe untersuchen */
          for(j=0;j<3;j++)
          {
               /* Ist diese ZK länger als ihr Nachfolger ? */
               if ( strlen(name[i][j]) > strlen(name[i][j+1]) )
               {
                    /* Tauschen */
                    strcpy(temp,name[i][j]);
                    strcpy(name[i][j],name[i][j+1]);
                    strcpy(name[i][j+1],temp);
                    getauscht = 1;
               }
          }
     }
     while(getauscht);

/* Ausgabe der beiden Reihen von Zeichenketten */
printf("\n");
for(i=0;i<2;i++)
{
     for(j=0;j<4;j++)
     {
          printf("%s",name[i][j]);

          /* Auffüllen mit Leerstellen */
```

```
                leerstellen = 12 - strlen(name[i][j]);
                for(k=0;k<leerstellen;k++)
                    printf(" ");
            }
            printf("\n");
        }
}
```

20.13 Zu Kapitel 13

Lösung zu Übung U1301:

```
/* Datei U1301.c */
#include <stdio.h>

void main(void)
{
        /* sizeof liefert eine ganze Zahl */
        printf("\n%i",sizeof(char));
        printf("\n%i",sizeof(int));
        printf("\n%i",sizeof(long int));
        printf("\n%i",sizeof(float));
        printf("\n%i",sizeof(double));
        printf("\n%i",sizeof(long double));
}
```

Lösung zu Übung U1302:

- Es gibt insgesamt vier Funktionen: main(), f1(), f2() und f3()
- Die Variable z gibt es dreimal: als globale Variable und zweimal als lokale Variable in den Funktionen main() und f2()
- Die zu main() lokale Variable wird initialisiert, ausgegeben und erhöht
- Die globale Variable wird in f3() ausgegeben und erhöht
- Die zu f2() lokale Variable wird initialisiert, ausgegeben und erhöht
- Die zu main() lokale Variable wird an f1() übergeben, dort ausgegeben und erhöht

Lösung zu Übung U1303:

- d*b ergibt Fließkomma-Zahl 12.5, f wird in Int-Zahl 5 umgewandelt, Ergebnis der Division ist wieder Fließkomma-Zahl 2.5, diese wird an e übergeben
- b wird in double-Zahl 5.0 umgewandelt, Ergebnis der Division ist die double-Zahl 0.5, Ergebnis der Multiplikation ist die Fließkomma-Zahl 1.0, diese wird bei der Übergabe an c in die Ganzzahl 1 umgewandelt

- Ergebnis der Division ist die Fließkomma-Zahl 2.5, diese wird ausgegeben
- e wird in Ganzzahl 2 verwandelt, Ergebnis der Division ist die Ganzzahl 1, diese wird ausgegeben

20.14 Zu Kapitel 14

Lösung zu Übung U1402:

```
/* Datei U1402.c */
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
       int i;
       char s[5], t[5];

       for(i=1;i<=20;i++)
       {
            itoa(i,s,10);        /* Umwandeln der Zahl in Zahl-Zeichenkette */
            strcpy(t,"0");      /* Beginn der Ausgabe-Zeichenkette */

            /* Falls die Zahl einstellig ist, also die Zahl-Zeichenkette */
            /* nur ein Zeichen umfaßt, eine weitere Null an die Ausgabe- */
            /* Zeichenkette anhängen                                      */
            if (strlen(s)==1) strcat(t,"0");

            /* Zahl-Zeichenkette an Ausgabe-Zeichenkette anhängen */
            strcat(t,s);

            printf("%10s",t);         /* Ausgabe */
       }
}
```

Lösung zu Übung U1403:

```
/* Datei U1403.c */
#include <stdio.h>
#include <stdlib.h>

void main(void)
{
       int i;
       char s1[20], s2[20], s3[20], s4[20];
```

```
        /* Tabellen-Überschrift */
        printf("\n Basis 2     Basis 8   Basis 10   Basis 16\n");

        /* Tabelle */
        for(i=1;i<=20;i++)
        {
             itoa(i,s1,2);            /* Umwandlung mit Basis 2 */
             itoa(i,s2,8);            /* Umwandlung mit Basis 8 */
             itoa(i,s3,10);           /* Umwandlung mit Basis 10 */
             itoa(i,s4,16);           /* Umwandlung mit Basis 16 */

             printf("%10s%10s%10s%10s\n",s1,s2,s3,s4);
        }
}
```

Lösung zu Übung U1405:

```
/* Datei U1405.c */
#include <stdio.h>
void main(void)
{
        /* Eine int-Variable, 2 Zeiger auf int-Variablen */
```

```
        int a=23, *pa, *pb;

        /* pa bekommt die Adresse von a oder "pa zeigt auf a" */
        pa=&a;

        /* Inhalt von pa wird ausgegeben */
        printf("%i\n",*pa);

        /* Inhalt von pa wird um 10 vermindert */
        *pa=*pa-10;

        /* pb zeigt auf das gleiche wie pa */
        pb=pa;

        /* Inhalt von pb wird ausgegeben */
        printf("%i\n",*pb);
}
```

Lösung zu Übung U1406:

```
/* Datei U1406.c */
#include <stdio.h>
void main(void)
{
        /* Ein initialisiertes Feld mit 8 Elementen, 2 Zeiger */
        int a[8]={2,5,4,6,11,54,76,23}, i, *pa, *pb;

        for (i=0;i<8;i++)
        {
             pa=&a[i];          /* pa zeigt auf a[i], Rest s.o. */
             printf(" %5i",*pa);
             *pa=*pa-10;
             pb=pa;
             printf(" %5i\n",*pa);
        }
}
```

Lösung zu Übung U1408:

```
/* Datei U1408.c */
#include <stdio.h>

/* Prototyp der Tausch-Funktion, Parameter sind zwei Zeiger auf double */
```

```
void swap(double *, double *);

void main(void)
{
       double a,b,*pa,*pb;

       /* Die Variablen bekommen Werte zugewiesen */
       a=7.3/3.0;
       b=23.3/7.0;

       /* Die Zeiger bekommen die Adressen der Variablen zugewiesen */
       pa=&a;
       pb=&b;

       printf("%lf %lf\n",a,b);                /* Werte vorher */

       /* Tausch-Funktion aufrufen, Parameter sind die beiden Zeiger */
       swap(pa,pb);

       printf("%lf %lf\n",a,b);                /* Werte nachher */

}
/* Tausch-Funktion, Parameter sind zwei Zeiger auf double */
void swap(double *px, double *py)
{
       double temp;

       /* Tauschen der Inhalte der beiden Zeiger. Das Tauschen der Zeiger */
       /* hätte nur Auswirkung in der Fkt., aber nicht im Hauptprogramm!  */
       temp=*px;
       *px=*py;
       *py=temp;
}
```

Lösung zu Übung U1409:

```
/* Datei U1409.c */
#include <stdio.h>
void hoch(double, double *, double *, double *);

void main(void)
{
       double x, x2, x3, x4, *p2, *p3, *p4;
```

```
        x=3.4;

        /* Die drei Zeiger müssen auf Adressen zeigen! */
        p2 = &x2;
        p3 = &x3;
        p4 = &x4;

        /* Funktionsaufruf */
        hoch(x,p2,p3,p4);

        /* Ausgabe */
        printf("\n%lf",x);
        printf("\n%lf",x2);
        printf("\n%lf",x3);
        printf("\n%lf",x4);
}
void hoch(double t, double *t2, double *t3, double *t4)
{
        /* Berechnung der Ergebnisse und Zuweisung an die Adressen, auf    */
        /* die gezeigt wird (lokal: Inhalt von t2 bis t4, in main: Inhalt */
        /* von p2 bis p4                                                   */
        *t2 = t * t;
```

```
        *t3 = *t2 * t;
        *t4 = *t3 * t;
}
```

20.15 Zu Kapitel 15

Lösung zu Übung U1502:

```
/* Datei U1502.c */
#include <stdio.h>
void main(void)
{
        double x[4]={3.2,5.4,7.6,9.8}, *pa;
        int i;

        /* Lösung 1: Adressen und Werte der Feld-Elemente */
        printf("\n");
        for (i=0; i<4; i++)
             printf("%i: %lf ",&x[i],x[i]);

        /* Lösung 2: */
        printf("\n");
        for (i=0; i<4; i++)
        {
             /* Zeiger zeigt auf Feld-Element i */
             pa = &x[i];

             /* Adresse und Inhalt des Zeigers */
             printf("%i: %lf ",pa,*pa);
        }

        printf("\n");

        /* Lösung 3: */
        pa=x;                                  /* Zeiger zeigt auf Feld-Anfang */
        for (i=0; i<4; i++)
        {
             printf("%i: %lf ",pa,*pa);
             pa++;        /* Zeiger wird verschoben */
        }

        printf("\n");

        /* Lösung 4: Feldname wird als Zeiger interpretiert */
```

```
        for (i=0; i<4; i++)
             /* Adresse und Inhalt des Zeigers */
             printf("%i: %lf ",x+i,*(x+i));
}
```

Lösung zu Übung U1504:

```
/* Datei U1504.c */
#include <stdio.h>

/* An f1 wird die Feldadresse, die Größe des Feldes und die drei */
/* Adressen der zu ermittelnden Werte übergeben                  */
void f1(double *,int, double *, double *, double *);

/* An inv werden die Feldadresse von Originalfeld und umgekehrtem */
/* Feld und die gemeinsame Größe der beiden Felder übergeben          */
void inv(double *, double *, int);

/* An sort wird die Feldadresse und die Größe des Feldes übergeben */
void sort(double *, int);

void main(void)
{
        double x[4]={11.2,8.4,1.6,12.8}, y[4];
        double summe, mittelwert, maximum;
        int i;

        /* Ausgabe vor der Bearbeitung */
        printf("\nFeld1: ");
        for (i=0;i<4;i++)
             printf("%lf ",x[i]);

        /* Aufruf der drei Funktionen mit Feldadressen, */
        /* Größe und Adressen der Rückgabewerte         */
        f1(x,4,&summe,&mittelwert,&maximum);
        inv(x,y,4);
        sort(x,4);

        /* Ausgabe der Rückgabewerte */
        printf("\nSumme    : %lf",summe);
        printf("\nMittelwert: %lf",mittelwert);
        printf("\nMaximum     : %lf",maximum);
```

```
        /* Ausgabe der beiden Felder */
        printf("\nFeld2: ");
        for (i=0;i<4;i++)
             printf("%lf ",y[i]);
        printf("\nFeld1: ");
        for (i=0;i<4;i++)
             printf("%lf ",x[i]);
}

/* Funktion f1() */
void f1(double *feld, int size, double *sum,
        double *mit, double *max)
{
        int i;

        /* Speicherung der ermittelten Werte direkt */
        /* über die übergebenen Adressen */
        *sum = 0;
        *max = feld[0];
```

```
        for (i=0;i<size;i++)
        {
             *sum = *sum + feld[i];
             if (feld[i] > *max) *max = feld[i];
        }
        *mit = *sum/size;
}

/* Funktion inv() */
void inv(double *feld1, double *feld2, int size)
{
        int i;

        /* Index des invertierten Feldes entspricht Größe des      */
        /* Original-Feldes minus Index des Original-Feldes minus 1 */
        for (i=0;i<size;i++)
             feld2[size-i-1] = feld1[i];
}

/* Funktion sort() */
void sort(double *feld, int size)
{
        int getauscht, i;
        double temp;

        /* Solange Feld-Elemente durchgehen, bis bei einem */
        /* Durchgang nicht mehr getauscht wurde             */
        do
        {
             getauscht = 0;    /* bisher nicht getauscht */

             for(i=0;i<size-1;i++)
                  if (feld[i] < feld[i+1])
                  {
                       temp = feld[i+1];    /* tauschen */
                       feld[i+1] = feld[i];
                       feld[i] = temp;
                       getauscht = 1;
                  }
        }
        while(getauscht==1);
}
```

Lösung zu Übung U1505:

```
/* Datei U1505.c */
#include <stdio.h>
#include <string.h>
int suchen(char *, char);

void main(void)
{
       char t[40]={"Beliebige Zeichenfolge"};
       char x='e';

       /* Aufruf der Funktion und Ausgabe des */
       /* zurückgegebenen Such-Ergebnisses      */
       printf("Zeichen: %c in Text: %s: %i mal",x,t,suchen(t,x));
}

/* Parameter: Zeiger auf Zeichenkette und Such-Zeichen */
int suchen(char *t, char x)
{
       int i, anz=0;
       for (i=0;i<strlen(t);i++)
            /* Zähler erhöhen, falls Such-Zeichen gefunden wurde */
            if (t[i]==x) anz++;
       return(anz);                        /* Zähler zurückgeben */
}
```

Lösung zu Übung U1507:

```
/* Datei U1507.c */
#include <stdio.h>
#include <string.h>
void suchen(char u[7][12], char y, int *anzahl);

void main(void)
{
       char t[7][12]={{"Montag"},{"Dienstag"},{"Mittwoch"},
            {"Donnerstag"},{"Freitag"},{"Samstag"},{"Sonntag"}};
       char x='e';
       int anz[7], i;

       /* Aufruf der Funktion, Parameter: Zeiger auf Zeichenkette */
       /* Such-Zeichen und Zeiger auf Zählerfeld                  */
       suchen(t,x,anz);
```

```
        /* Ausgabetabelle mit allen Zählern */
        for (i=0;i<7;i++)
             printf("\nZeichen: %c in Text: %s: %i mal",
                  x,t[i],anz[i]);
}

void suchen(char u[7][12], char y, int *anzahl)
{
        int i, k;

        /* Jeweiligen Zähler erhöhen, falls Such-Zeichen gefunden wurde */
        for (i=0;i<7;i++)
        {
             anzahl[i]=0;                    /* Zähler Initialisieren */
             for (k=0;k<strlen(u[i]);k++)
                  if (u[i][k]==y) anzahl[i]++;
        }
}
```

Lösung zu Übung U1508:

```
/* Datei U1508.c */
#include <stdio.h>
#include <string.h>
int pos(char *, char *);

void main(void)
{
       /* Text, in dem gesucht wird */
       char t[40]={"Beliebige Zeichenfolge"};

       /* Such-Zeichenfolge */
       char x[5]={"eli"};

       /* Aufruf der Funktion und Ausgabe des */
       /* zurückgegebenen Such-Ergebnisses      */
       printf("Zeichenfolge: %s in Text: %s an Position %i",
                x,t,pos(t,x));
}

int pos(char *t, char *x)
{
       int i, j, gefunden=0;

       /* Suchen des ersten Zeichens der Such-Zeichenfolge */
       for (i=0;i<strlen(t);i++)
       {
            /* Falls erstes Zeichen gefunden wurde: */
            if (t[i]==x[0])
            {
                 gefunden = 1;

                 /* Vergleich der weiteren Zeichen */
                 for (j=1;j<strlen(x);j++)

                      /* Falls ein weiteres Zeichen nicht übereinstimmt  */
                      /* oder das Ende des Textes erreicht wurde,       */
                      /* in dem gesucht wird, wird abgebrochen.         */
                      if (t[i+j]!=x[j] || t[i+j]=='\0')
```

```
                    {
                        gefunden=0;
                        break;
                    }
            }

            /* Falls nicht abgebrochen wurde: Such-Zeichenfolge */
            /* wurde an Position i gefunden, Rücksprung         */
            if (gefunden==1) return(i);
        }

        /* Falls Such-Zeichenfolge gar nicht gefunden wurde: */
        /* Rücksprung mit "Position" -1                      */
        return(-1);
}
```

Lösung zu Übung U1509:

```
/* Datei U1509.c */
#include <stdio.h>
#include <string.h>
void drehe(char *, char *);

void main(void)
{
        char t[40]={"Beliebige Zeichenfolge"};
        char u[40];
        printf("\n%s",t);
        drehe(t,u);
        printf("\n%s",u);
}

void drehe(char *a, char *b)
{
        int i;

        /* Index der invertierten Zeichenkette entspricht Länge der          */
        /* Original-Zeichenkette minus Index des Original-Feldes minus 1 */
        for (i=0;i<strlen(a);i++)
            b[strlen(a)-i-1] = a[i];

        /* Anhängen des Ende-Zeichens erforderlich */
```

```
        b[strlen(a)]='\0';
}
```

Lösung zu Übung U1510:

```
/* Datei U1510.c */
#include <stdio.h>
void mw(double u[2][2], double v[2]);
void main(void)
{
        double x[2][2]={{11.2,8.4},{1.6,12.8}}, mit[2];
        int i, j;

        /* Funktionsaufruf mit zwei Zeigern auf Feldbeginn */
        mw(x,mit);

        /* Ergebnis-Tabelle */
        for (i=0;i<2;i++)
        {
             for (j=0;j<2;j++)
                  printf("%10.3lf ",x[i][j]);
             printf("    Mittelwert: %10.3lf \n",mit[i]);
```

```
        }
}

void mw(double u[2][2], double v[2])
{
        int i, j;
        for (i=0;i<2;i++)
        {
             /* Einzeln Summen bzw. Mittelwerte initialisieren */
             v[i] = 0;

             /* Summen der Elemente einer Zeile berechnen */
             for (j=0;j<2;j++)
                  v[i]+=u[i][j];

             /* Summen in Mittelwerte "umwandeln" */
             v[i] /= 2;
        }
}
```

20.16 Zu Kapitel 16

Lösung zu Übung U1603:

```
/* Datei U1603.c */
#include <stdio.h>
void main(void)
{
        FILE *dp;
        char z[200];
        int i = 0, lg, ziffern, gesamt = 0, zeilen = 0;
        dp = fopen("U1601.TXT","r");

        if (dp != NULL)
        {
             while(feof(dp) == 0)
             {
                  ziffern = 0;
                  fgets(z, 200, dp);
                  lg = strlen(z);
                  for (i=0; i<lg; i++)
                       if (z[i] >= 48 && z[i] <= 57)
                            ziffern++;
```

```
                gesamt += ziffern;
                printf("\nZeile: %3i, Anzahl: %3i",zeilen,ziffern);
                zeilen++;
            }

            fclose(dp);
            printf("\nZeilen insgesamt: %3i",zeilen);
            printf("\nAnzahl insgesamt: %3i",gesamt);
        }
        else
            printf("\nDatei nicht ge"ffnet");
}
```

Lösung zu Übung U1605:

```
/* Datei U1605.c */
#include <stdio.h>
void main(void)
{
        FILE *dpe, *dpa, *dpb;
        char z[200];
        dpe = fopen("U1605.TXT","r");
        dpa = fopen("U1605A.TXT","w");
        dpb = fopen("U1605B.TXT","w");

        if (dpe != NULL && dpa != NULL && dpb != NULL)
        {
            while(feof(dpe)==0)
            {
                fgets(z,200,dpe);
                if (z[0]=='#' || z[1]=='#' || z[2]=='#')
                    fprintf(dpa,"%s",z);
                else
                    fprintf(dpb,"%s",z);
            }

            fclose(dpe);
            fclose(dpa);
            fclose(dpb);
        }
        else
            printf("Fehler\n");
}
```

Lösung zu Übung U1609:

```
/* Datei U1609.c */
#include <stdio.h>
#include <stdlib.h>

void lesen(FILE *, int);
void aendern(FILE *, int);
void hinzu(FILE *, int *);

void main(void)
{
        int anzahl, auswahl;
        char anztext[30];

        FILE *dp;
        dp = fopen("U1609.txt","r+");

        if (dp != NULL)
        {
             /* Anzahl Datensätze */
             fgets(anztext,10,dp);
```

```
            anzahl = atoi(anztext);
            printf("\nAnzahl der Datensaetze: %i\n",anzahl);

            do
            {
                printf("\nMenu: ");
                printf("\n1: Lesen aller Datensaetze");
                printf("\n2: Veraendern eines Datensatzes");
                printf("\n3: Hinzufuegen eines Datensatzes");
                printf("\n4: Programm beenden");
                printf("\n\nIhre Auswahl: ");
                scanf("%i",&auswahl);

                switch (auswahl)
                {
                    case 1: lesen(dp, anzahl); break;
                    case 2: aendern(dp, anzahl); break;
                    case 3: hinzu(dp, &anzahl); break;
                    case 4: printf("\nProgramm beendet\n"); break;
                    default: printf("\n Auswahl nur 1-4");
                }
            }
            while(auswahl!=4);
        }
        else
            printf("Datei nicht gefunden");

        /* Schliessen, falls geöffnet */
        if (dp != NULL)
            fclose(dp);
}

/* Alle Datensätze lesen und ausgeben */ void lesen(FILE *dp, int anzahl)
{
        int i;
        char dsatz[30];
        printf("\nAlle %i Datensaetze:\n",anzahl);
        /* In Datensatz 0 steht die Anzahl */
        for(i=1; i<=anzahl; i++)
        {
            fseek(dp,i*22,SEEK_SET);
            fgets(dsatz,30,dp);
```

```
            printf("%2i: %s",i,dsatz);
        }
}

/* Datensatz auswählen und ändern */
void aendern(FILE *dp, int anzahl)
{
        int nummer;
        char dsatz[30];
        printf("\nVeraendern des Datensatzes Nr. (1 - %i): ",anzahl);
        scanf("%i",&nummer);

        /* Datensatz ändern, Tabelle ausgeben */
        if (nummer > 0 && nummer <= anzahl)
        {
            printf("Neuer Eintrag: ");
            scanf("%s",dsatz);
            fseek(dp,nummer*22,SEEK_SET);
            fprintf(dp,"%-20s",dsatz);
            lesen(dp, anzahl);
        }
        else
            printf("Nur 1 bis %i eingeben !\n",anzahl);
}


/* Beim Hinzufügen wird die geänderte Anzahl auch zurückgegeben */
void hinzu(FILE *dp, int *p_anzahl)
{
        int nummer;
        char anztext[30], dsatz[30];

        /* Neue Anzahl ermitteln und in die erste Zeile schreiben */
        *p_anzahl = *p_anzahl + 1;
        itoa(*p_anzahl,anztext,10);
        fseek(dp,0,SEEK_SET);
        fprintf(dp,"%-20s",anztext);

        /* Datensatz hinzufügen, Tabelle ausgeben */
        printf("\nHinzufuegen eines Datensatzes:\n");
        printf("Neuer Eintrag: ");
        scanf("%s",dsatz);
```

```
        fseek(dp,*p_anzahl*22,SEEK_SET);
        fprintf(dp,"%-20s\n",dsatz);
        lesen(dp, *p_anzahl);
}
```

20.17 Zu Kapitel 17

Lösung zu Übung U1702:

```
/* Datei U1702.c */
#include <stdio.h>
#include <string.h>
void main(void)
{
        struct adr
        {
             char strasse[30];
             int hausnr;
             long int plz;
             char ort[20];
        };

        /* Deklaration mehrerer Variablen dieses Strukturtyps */
```

```
struct adr adra, adrb, temp;

/* Zuweisung der Komponenten-Inhalte */
strcpy(adra.strasse,"Peterstraße");
adra.hausnr = 36;
adra.plz = 52074;
strcpy(adra.ort,"Aachen");

strcpy(adrb.strasse,"Ottostraße");
adrb.hausnr = 48;
adrb.plz = 52134;
strcpy(adrb.ort,"Herzogenrath");

/* Ausgabe vor dem Tauschvorgang */
printf("\n1: %s ",adra.strasse);
printf("%i ",adra.hausnr);
printf("in %li ",adra.plz);
printf("%s",adra.ort);

printf("\n2: %s ",adrb.strasse);
printf("%i ",adrb.hausnr);
printf("in %li ",adrb.plz);
printf("%s",adrb.ort);

/* Tauschvorgang, gesamte Struktur-Variable kann zugewiesen werden */
temp=adra;
adra=adrb;
adrb=temp;

/* Ausgabe nach dem Tauschvorgang */
printf("\n1: %s ",adra.strasse);
printf("%i ",adra.hausnr);
printf("in %li ",adra.plz);
printf("%s",adra.ort);

printf("\n2: %s ",adrb.strasse);
printf("%i ",adrb.hausnr);
printf("in %li ",adrb.plz);
printf("%s",adrb.ort);
```

Lösung zu Übung U1704:

```
/* Datei U1704.c */
#include <stdio.h>
#include <string.h>
void main(void)
{
       struct adr
       {
            char strasse[30];
            int hausnr;
            long int plz;
            char ort[20];
       };

       struct mitarbeiter
       {
            char vorname[20];
            char nachname[20];
            struct adr adra;
            float gehalt;
       };

       /* Deklaration eines Feldes von Variablen des Ober-Strukturtyps */
       struct mitarbeiter ps[10];
       int i;

       for (i=0;i<10;i++)
       {
            /* Zuweisung und Ausgabe einzelner Komponenten-Inhalte */
                         /* für alle Feld-Elemente                */
            ps[i].adra.hausnr = (i*5)+50;
            ps[i].gehalt = (i*20)+3500;
            printf("%3i %8.2f \n", ps[i].adra.hausnr, ps[i].gehalt);
       }
}
```

Lösung zu Übung U1705:

```
/* Datei U1705.c */
#include <stdio.h>
#include <string.h>

/* Globale Definition der Struktur,                    */
/* da sie in mehreren Funktionen genutzt wird          */
struct adr
{
       char strasse[30];
       int hausnr;
       long int plz;
       char ort[20];
};

void einles(struct adr *z1);
void zuweis(struct adr *z1);
void tausch(struct adr *z1, struct adr *z2);
void ausgeb(struct adr s1);

void main(void)
```

```
{
        struct adr adra, adrb;
        einles(&adra);              /* Einlesen von adra */
        zuweis(&adrb);              /* Zuweisen von adrb */
        ausgeb(adra);               /* Ausgeben vor dem Tauschvorgang */
        ausgeb(adrb);
        tausch(&adra,&adrb);        /* Tauschvorgang */
        ausgeb(adra);               /* Ausgeben nach dem Tauschvorgang */
        ausgeb(adrb);
}

/* Parameter der Funktion einles() ist Zeiger auf Struktur-Variable      */
/* Dadurch wird erreicht, dass das Ergebnis der Eingabe erhalten bleibt */
void einles(struct adr *z1)
{
        printf("\nStraße: ");
        scanf("%s",&(*z1).strasse);
        printf("Hausnummer: ");
        scanf("%i",&(*z1).hausnr);
        printf("Postleitzahl: ");
        scanf("%li",&(*z1).plz);
        printf("Ort: ");
        scanf("%s",&(*z1).ort);
}

void zuweis(struct adr *z1)
{
        strcpy(z1->strasse,"Ottostraße");
        z1->hausnr = 48;
        z1->plz = 52134;
        strcpy(z1->ort,"Herzogenrath");
}

/* Inhalte der Zeiger, also Struktur-Variablen werden getauscht */
void tausch(struct adr *z1, struct adr *z2)
{
        struct adr temp;
        temp = *z1;
        *z1 = *z2;
        *z2 = temp;
}
```

```
void ausgeb(struct adr s1)
{
       printf("\n%s ",s1.strasse);
       printf("%i ",s1.hausnr);
       printf("in %li ",s1.plz);
       printf("%s",s1.ort);
}
```

Lösung zu Übung U1706:

```
/* Datei U1706.c */
#include <stdio.h>

struct temperatur
{
       double x;
       double y;
       double z;
       double wert;
};

/* Eingabe-Ergebnis bleibt erhalten, da die Adresse des      */
/* Feldes per Zeiger übergeben wurde                         */
void tein(struct temperatur p[5])
{
       int i;
       for (i=0;i<5;i++)
       {
            printf("\nElement %i",i);
            printf("\nx: ");
            scanf("%lf",&p[i].x);
            printf("y: ");
            scanf("%lf",&p[i].y);
            printf("z: ");
            scanf("%lf",&p[i].z);
            printf("wert: ");
            scanf("%lf",&p[i].wert);
       }
}

void tsort(struct temperatur p[5])
{
```

```
        /* Getauscht werden die Feldelemente, also Struktur-Variablen,   */
        /* es wird zusätzlich eine einzelne Struktur-Variable benötigt   */
        struct temperatur a;

        int getauscht, i;
        do
        {
             getauscht = 0;
             for (i=0;i<4;i++)
                  /* Tausch-Kriterium: Temperatur-Wert */
                  if (p[i].wert > p[i+1].wert)
                  {
                       getauscht = 1;
                       a = p[i];
                       p[i] = p[i+1];
                       p[i+1] = a;
                  }
        }
        while (getauscht);
}

void taus(struct temperatur p[5])
{
        int i;
        for (i=0;i<5;i++)
             printf("\nElement %i xyz: %6.2lf %6.2lf %6.2lf Wert: %6.2lf",
                  i, p[i]);
}

void main(void)
{
        struct temperatur tfeld[5];
        tein(tfeld);
        tsort(tfeld);
        taus(tfeld);
}
```

20.18 Zu Kapitel 18

Lösung zu Übung U1802:

```
/* Datei U1802.c */
#include <stdio.h>
```

```
#include <stdlib.h>
void main(void)
{
       int *zahl;
       int i;

       /* Speicherplatz-Anforderung für 15 int-Zahlen */
       zahl = (int *) malloc(15 * sizeof(int));

       if (zahl == NULL)
       {
            printf("Nicht genügend Speicher!\n");
            exit(0);
       }

       /* Nicht initialisierte Inhalte anzeigen */
       for (i=0;i<15;i++)
            printf("%3i ",zahl[i]);
       printf("\n");

       /* Inhalte initialisieren und anzeigen */
       for (i=0;i<15;i++)
       {
            zahl[i] = 333 + 5*i;
            printf("%3i ",zahl[i]);
       }

}
```

Lösung zu Übung U1803:

```
/* Datei U1803.c */
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
       double *zahl;
       int i;

       /* Speicherplatz-Anforderung für 5 double-Zahlen */
       zahl = (double *) malloc(5 * sizeof(double));

       if (zahl == NULL)
```

```
        {
             printf("Nicht genügend Speicher!\n");
             exit(0);
        }

        /* Nicht initialisierte Inhalte anzeigen */
        for (i=0;i<5;i++)
             printf("%6.2lf ",zahl[i]);
        printf("\n");

        /* Inhalte initialisieren und anzeigen */
        for (i=0;i<5;i++)
        {
             zahl[i] = 3.3 + 5*i;
             printf("%6.2lf ",zahl[i]);
        }

        /* Speicherplatz am gewünschter Stelle des Programmes freigeben */
        free(zahl);
}
```

Lösung zu Übung U1804:

```
/* Datei U1804.c */
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
        double *zahl;
        int i, anz;

        /* Wieviele double-Zahlen sollen gespeichert werden */
        printf("Wie viele double-Zahlen ? ");
        scanf("%i",&anz);

        /* Speicherplatz-Anforderung mit calloc */
        zahl = (double *) calloc(anz, sizeof(double));

        if (zahl == NULL)
        {
             printf("Nicht genügend Speicher!\n");
             exit(0);
```

```
        }

        /* Mit Wert 0 initialisierte Inhalte anzeigen */
        for (i=0;i<anz;i++)
             printf("%6.2lf ",zahl[i]);
        printf("\n");

        /* Inhalte neu belegen und anzeigen */
        for (i=0;i<anz;i++)
        {
             zahl[i] = 3.3 + 5*i;
             printf("%6.2lf ",zahl[i]);
        }

        free(zahl);
}
```

Lösung zu Übung U1805:

```
/* Datei U1805.c */
#include <stdio.h>
#include <stdlib.h>
void main(void)
{
        double *zahl;
        int anz, i;

        /* Erste Speicheranforderung, Belegung und Ausgabe */
        printf("1) Wie viele double-Zahlen ? ");
        scanf("%i",&anz);

        zahl = (double *) calloc(anz, sizeof(double));

        if (zahl == NULL)
        {
             printf("Nicht genügend Speicher!\n");
             exit(0);
        }

        printf("%i Zahlen\n",anz);
        for (i=0;i<anz;i++)
             printf("%6.2lf ",zahl[i]);
```

```
        printf("\n");
        for (i=0;i<anz;i++)
        {
             zahl[i] = 3.3 + 5*i;
             printf("%6.2lf ",zahl[i]);
        }


        /* Zweite Speicheranforderung, Belegung und Ausgabe */
        printf("\n\n2) Wie viele double-Zahlen ? ");
        scanf("%i",&anz);

        zahl = (double *) realloc(zahl, anz * sizeof(double));

        if (zahl == NULL)
        {
             printf("Nicht genügend Speicher!\n");
             exit(0);
        }

        printf("%i Zahlen\n",anz);
        for (i=0;i<anz;i++)
             printf("%6.2lf ",zahl[i]);

        printf("\n");
        for (i=0;i<anz;i++)
        {
             zahl[i] = 3.3 + 5*i;
             printf("%6.2lf ",zahl[i]);
        }

        free(zahl);
}
```

Lösung zu Übung U1806:

```
/* Datei U1806.c */
#include <stdio.h>
#include <stdlib.h>

struct adr
{
```

```
        char strasse[30];
        int hausnr;
        long int plz;
        char ort[20];
};

void main(void)
{
        int i, anz;
        struct adr *padr;
        FILE *ein;

        /* Öffnet Eingabedatei zum Lesen */
        ein = fopen("u18ein.dat","r");

        if (ein!=NULL)
        {
             /* In der 1. Zeile sollte die Anzahl der Datensätze stehen */
             fscanf(ein,"%i",&anz);

             /* Der entsprechende Speicherplatz wird reserviert */
             padr = (struct adr *) malloc(anz * sizeof(struct adr));

             /* Die angegebene Anzahl von Datensätzen wird eingelesen */
             for(i=0;i<anz;i++)
             {
                  fscanf(ein,"%s",padr[i].strasse);
                  fscanf(ein,"%i",&padr[i].hausnr);
                  fscanf(ein,"%li",&padr[i].plz);
                  fscanf(ein,"%s",padr[i].ort);
             }

             /* Danach kann die Eingabedatei sofort geschlossen werden */
             fclose(ein);

             /* Die angegebene Anzahl von Datensätzen wird ausgegeben */
             for(i=0;i<anz;i++)
             {
                  printf("\n%s ",padr[i].strasse);
                  printf("%i in ",padr[i].hausnr);
                  printf("%li ",padr[i].plz);
                  printf("%s",padr[i].ort);
```

```
            }
            free(padr);
        }
        else
        {
            printf("Eingabedatei konnte nicht geöffnet werden");
            exit(0);
        }
}
```