

Prof. Dr. Helmut Herold
Prof. Dr. Bruno Lurz
Prof. Dr. Michael Zwanger

Übungen zur Programmiersprache C

**für das Praktikum
zu Informatik 1**

Inhaltsverzeichnis

0	Einleitung	1
0.1	Informationen zu den einzelnen Übungen	1
1	Einführendes Beispiel	3
1.1	Ausgabe eines Menüs <i>C-Syntax(1;<20)</i>	3
2	Elementare Datentypen	5
2.1	Alle möglichen Datentyp-Angaben in C <i>C-Syntax(4;<20)</i>	5
2.2	Bereichsüberläufe beim Datentyp short <i>C-Syntax(4;<10)</i>	5
3	Konstanten	7
3.1	Bitmuster beim Datentyp char <i>C-Syntax(3;<10)</i>	7
3.2	Erlaubte und unerlaubte Gleitpunktkonstanten <i>C-Syntax(3;<20)</i>	7
4	Variablen	9
4.1	Erlaubte und unerlaubte Variablennamen <i>C-Syntax(2;<20)</i>	9
5	Ausdrücke und Operatoren	11
5.1	Der einfache Zuweisungsoperator =	11
5.1.1	Zuweisen von unterschiedlichen Konstanten <i>C-Syntax(3;<10)</i>	11
5.2	Arithmetische Operatoren	11
5.2.1	Umformen mathematischer Ausdrücke in C <i>C-Syntax(4;<10)</i>	11
5.2.2	Kettenbruchentwicklung von PI <i>Mathematik(4;<10)</i>	12
5.4	Logische Operatoren	13
5.4.1	Überprüfungen mit logischen Operatoren <i>C-Syntax(3;<10)</i>	13
5.5	Bit-Operatoren	14
5.5.1	Überprüfungen mit Bit-Operatoren <i>Programmierung(4;<10)</i>	14

5.6	Shift-Operatoren		15
5.6.1	Vertauschen von zwei Bytes	<i>Programmierung(4;<10)</i>	15
5.6.2	Duale Ausgabe eines Bytes	<i>Programmierung(3;<20)</i>	15
6	Symbolische Konstanten		17
6.1	Konstanten-Definition mit #define		17
6.1.1	Volumen und Oberfläche einer Kugel	<i>Mathematik(3;<20)</i>	17
6.1.2	Lichtjahre in Kilometer umrechnen	<i>Physik(3;<20)</i>	17
6.1.3	Das Phänomen der entfesselten Erde	<i>Allgemein(4;<20)</i>	18
6.2	Konstanten-Definition mit const		18
6.2.1	Fallzeit für einen Körper	<i>Physik(3;<20)</i>	18
6.2.2	Benzinverbrauch und Geschwindigkeit	<i>Allgemein(3;<50)</i>	19
6.2.3	Zinsertrag für ein bestimmtes Kapital	<i>Wirtschaft(3;<20)</i>	19
7	Ein- und Ausgabe		21
7.1	Headerdateien und #include		21
7.1.1	Umrechnung von Geschwindigkeiten	<i>Physik(3;<50)</i>	21
7.2	Ein- und Ausgabe eines Zeichens		22
7.2.1	Quersumme zu einer 5-stelligen Zahl	<i>Mathematik(4;<20)</i>	22
7.3	Die Ausgabe mit printf()		22
7.3.1	Ausgeben eines Rahmens	<i>Allgemein(6;<20)</i>	22
7.4	Die Eingabe mit scanf()		22
7.4.1	Dezimal- und Hexawert zu einer Oktalzahl	<i>C-Syntax(2;<10)</i>	22
8	Datentypumwandlungen		23
8.1	Implizite Datentypumwandlungen		23
8.1.1	Reißleine von Fallschirm wann ziehen?	<i>Physik(3;<20)</i>	23
8.2	Explizite Datentypumwandlungen		24
8.2.1	Prozentzahlen für Kandidaten bei einer Wahl	<i>Allgemein(3;<30)</i>	24
11	Die if-Anweisung		25
11.1	Schaltjahre	<i>C-Syntax(3;<30)</i>	25
11.2	Tageszeit abhängig grüssen	<i>Allgemein(3;<30)</i>	26
11.3	Zahl x durch Zahl y teilbar?	<i>Mathematik(2;<30)</i>	26
13	Die switch-Anweisung		27
13.1	Menü-Auswahl	<i>Programmierung(2;<40)</i>	27

13.2	Flächen zu geometrischen Figuren	<i>Mathematik(3;<60)</i>	28
13.3	Folgedatum zu einem Datum	<i>Allgemein(4;<60)</i>	29
13.4	Fläche, Umfang und Radius eines Kreises	<i>Mathematik(3;<60)</i>	30
15	Die for-Anweisung								31
15.1	Berechnung der harmonischen Reihe	<i>Mathematik(2;<20)</i>	31
15.2	Summe von ungeraden Zahlen	<i>Mathematik(2;<20)</i>	31
15.3	Berechnung der Leibniz-Reihe	<i>Mathematik(3;<20)</i>	32
15.4	Berechnung der Exponential-Reihe	<i>Mathematik(4;<30)</i>	32
15.5	Anzahl der Handschläge auf einer Party	<i>Allgemein(3;<20)</i>	33
15.6	Alle Teiler zu einer Zahl	<i>Mathematik(4;<30)</i>	33
16	Die while-Anweisung								35
16.1	Fritz und Hans essen Äpfel	<i>Allgemein(3;<40)</i>	35
16.2	Eingabe einer geraden Zahl zwischen 1 und 100	<i>Allgemein(3;<20)</i>	36
16.3	Verschlüsseln mit Verschiebchiffre	<i>Allgemein(3;<30)</i>	36
16.4	Minimum und Maximum von n Zahlen	<i>Allgemein(4;<50)</i>	37
16.5	Fibonacci-Zahlen	<i>Mathematik(4;<30)</i>	37
16.6	Quersumme einer Zahl	<i>Mathematik(4;<20)</i>	39
16.7	Duale BCD-Darstellung einer Zahl	<i>Mathematik(4;<50)</i>	39
17	Die do...while-Anweisung								41
17.1	Zahlen raten	<i>Allgemein(3;<50)</i>	41
17.2	Quadratwurzel mit NEWTON-Iteration	<i>Mathematik(4;<50)</i>	42
18	Die break-Anweisung								45
18.1	Würfelspiel Bis 100	<i>Allgemein(4;<110)</i>	45
18.2	Primzahlen	<i>Mathematik(4;<40)</i>	46
22	Funktionen								47
22.1	Allgemeines zu Funktionen		47
22.2	Erstellen eigener Funktionen		47
22.2.1	Länge eines Streckenzuges	<i>Mathematik(3;<50)</i>	47
22.2.2	Funktionen für binäre Operationen	<i>DV-Wissen(4;<150)</i>	48
22.3	Die Parameter von Funktionen		49
22.3.1	Zeiten-Taschenrechner	<i>Allgemein(4;<120)</i>	49
22.3.2	Sortieren von 4 Zahlen	<i>DV-Wissen(4;<70)</i>	49

24 Präprozessor-Direktiven	51
24.1 Konstanten in limits.h und float.h <i>C-Syntax(3;<70)</i>	51
25 Zeiger und Arrays	53
25.1 Eindimensionale Arrays	53
25.1.1 Ziehen der Lottozahlen simulieren <i>Allgemein(4;<40)</i>	53
25.1.2 Primzahlen mit dem Sieb des Eratosthenes <i>Mathematik(3;<20)</i>	54
25.1.3 Das Ziegenproblem <i>Wahrscheinlichkeitstheorie(4;<40)</i>	55
25.2 Mehrdimensionale Arrays	56
25.2.1 Matrizenmultiplikation <i>Mathematik(4;<100)</i>	56
25.2.2 Das Spiel Tictac <i>Allgemein(4;<150)</i>	58
25.2.3 Game of Life (Beispiel für zellulare Automaten) <i>Allgemein(5;<150)</i>	59

Kapitel 0

Einleitung

0.1 Informationen zu den einzelnen Übungen

Zu jeder Übung ist immer folgende Information angegeben, an der sich sofort folgendes erkennen lässt:

Ausgabe eines Menüs	<i>C-Syntax</i>	(1;	<20)
↑	↑	↑	↑
Überschrift	Fachgebiet	Schwierig- keitsgrad	Umfang

Wie zu sehen ist, werden neben der Überschrift noch drei weitere Informationen gegeben:

- ❑ Fachgebiet (C-Syntax, Allgemein, Wirtschaft, Mathematik usw.)
- ❑ Schwierigkeitsgrad
Die Schwierigkeitsgrade reichen dabei von 1 (sehr einfach) bis 6 (sehr schwer). Diese Festlegung des Schwierigkeitsgrades kann dabei natürlich nur subjektiv sein, da jede Person abhängig von ihrem Erfahrungs-, Bildungsstand und der momentanen Ideeneingebung die einzelnen Aufgaben unterschiedlich schwierig bzw. leicht empfinden wird.
- ❑ Umfang der Aufgabe
Hier wird in etwa der Umfang dieser Aufgabe in Zeilen angegeben. Diese Angabe ist auf die Lösungen bezogen und soll dem Leser einen Eindruck über den ungefähren Umfang der jeweiligen Aufgabenstellung geben. Natürlich kann die vom Leser erstellte Lösung mehr oder auch weniger Zeilen umfassen. Auf keinen Fall sollte der Leser versuchen, seine Lösung so umzuformen, dass das hier angegebene Kriterium (Zeilenzahl) erfüllt ist, da diese Information lediglich der Aufwandsbaschätzung im voraus dient.

Kapitel 1

Einführendes Beispiel

1.1 Ausgabe eines Menüs *C-Syntax(1;<20)*

Erstellen Sie ein C-Programm *menue.c*, das folgendes am Bildschirm ausgibt:

```
Hauptmenue
=====

(A) endern
(B) eenden
(D) rucken
(E) ingeben
(L) oeschen

Was wuenschen Sie zu tun ?
```

Kapitel 2

Elementare Datentypen

2.1 Alle möglichen Datentyp-Angaben in C *C-Syntax(4;<20)*

Zählen Sie alle in Standard-C möglichen Datentyp-Angaben auf! Dabei sollten Sie folgende Gruppeneinteilung für die einzelnen Datentypen vornehmen:

- ☐ Vorzeichenbehaftete Ganzzahltypen
- ☐ Nicht vorzeichenbehaftete Ganzzahltypen
- ☐ Gleitpunkttypen

2.2 Bereichsüberläufe beim Datentyp short *C-Syntax(4;<10)*

Hier wird wieder angenommen, dass der Datentyp **short** zwei Bytes belegt. Geben Sie nun die resultierende Dualdarstellung mit entsprechendem Dezimalwert für folgende Dezimalzahlen im **short**-Datentyp an:

-65000 (10) :	
100000 (10) :	
33000 (10) :	
65535 (10) :	

Kapitel 3

Konstanten

3.1 Bitmuster beim Datentyp `char` *C-Syntax(3;<10)*

Welches Bitmuster haben die folgenden `char`-Konstanten?

```
'%'
'?'
9
'g'
26
'{'
1245
```

3.2 Erlaubte und unerlaubte Gleitpunktkonstanten *C-Syntax(3;<20)*

Streichen Sie in folgender Liste alle in C nicht erlaubten Gleitpunktkonstanten:

```
2.333333333333e2531
3.4LF
.1234562772E15
.e+12
3.e-19L
2.3e+3.45
4444444.L
6365.F
2143,63
52.e++431
4.8e
```

Kapitel 4

Variablen

4.1 Erlaubte und unerlaubte Variablennamen *C-Syntax(2;<20)*

Streichen Sie in der folgenden Liste alle in C nicht erlaubten Variablennamen heraus:

```
hans_im_glueck
7_und_40_elf
____mittel_streifen
karl_iv
null_08
dreier*_hotel
abc_schuetze
kündigung
KINDERGARTEN
```

Kapitel 5

Ausdrücke und Operatoren

5.1 Der einfache Zuweisungsoperator =

5.1.1 Zuweisen von unterschiedlichen Konstanten C-Syntax(3;<10)

Welche dezimalen Werte hätten die jeweiligen Variablen nach den folgenden Zuweisungen? Begründen Sie Ihre Angaben und überprüfen Sie sie, indem Sie ein C-Programm schreiben, das Ihnen diese Variablen mit `printf("var=%d\n", var)` ausgibt.

```
short a=10;
short b=010;
short c='2';
short d=0x2;
short e=0x21;
unsigned short f=-9;
```

5.2 Arithmetische Operatoren

5.2.1 Umformen mathematischer Ausdrücke in C C-Syntax(4;<10)

Erstellen Sie ein Programm, das die folgenden beiden mathematischen Ausdrücke berechnet und jeweils das Ergebnis ausgibt. Klammern sollten dabei nur dann verwendet werden, wenn diese unbedingt notwendig sind.

$$\frac{18}{2} \cdot \frac{4+5}{9-6} \% \left(6 + \frac{8}{4}\right)$$

$$\frac{4-10 + \frac{100+100-40+80}{5 \cdot 2 \cdot 4} + 36}{\frac{90-30}{10-5}}$$

5.2.2 Kettenbruchentwicklung von PI *Mathematik(4;<10)*

Die Zahl π lässt sich auf sechs Stellen genau mit der folgenden Kettenbruchentwicklung berechnen:

$$\begin{aligned} \text{Pi} = 3 + & \cfrac{1}{7 + \cfrac{1}{15 + \cfrac{1}{1 + \cfrac{1}{292 + \cfrac{1}{2}}}}} \end{aligned}$$

Ergänzen Sie nun das folgende C-Programm *kettenpi.c*, indem sie diesen Kettenbruch in einen C-Ausdruck umwandeln, den Sie der Variablen `pi` zuweisen.

```
/*--- kettenpi.c -----*/
/*-----*/
#include <stdio.h>

int main(void)
{
    float pi;

    pi = .....

    printf("Pi = %f\n", pi);
    return(0);
}
```

Haben Sie `pi` den richtigen Kettenbruch zugewiesen, so sollte dieses Programm folgende Bildschirmausgabe liefern:

```
Pi = 3.141593
```

5.4 Logische Operatoren

5.4.1 Überprüfungen mit logischen Operatoren *C-Syntax(3;<10)*

Erstellen Sie ein Programm, das die Werte folgender Ausdrücke einer `int`-Variable `wert` zuweist und diese dann als Dezimalzahl, also mit

```
printf("n.Wert: %d\n", wert);
```

ausgibt.

-
1. ob der Wert der Variablen `a` im Intervall `[-20,100]` liegt:
(für `a = -15`)
 2. ob Wert der Variablen `x` negativ ist, aber zugleich auch der Wert der Variablen `y` im Intervall `[5,30]` liegt:
(für `x = 1` und `y = 7`)
 3. ob Wert der ganzzahligen Variablen `z` ungerade ist und zugleich auch durch 3 und 5 teilbar ist:
(für `z = 15`)
 4. ob der Wert der ganzzahligen Variablen `jahr`
 - durch 400 oder
 - durch 4, aber nicht durch 100 teilbar ist (Bedingung für ein Schaltjahr):(für `jahr = 2100`)
 5. ob das Produkt der beiden `int`-Variablen `a` und `b` in den Datentyp `unsigned char` ohne Überlauf untergebracht werden kann:
(für `a = 25` und `b = 10`)
 6. ob der Wert der `char`-Variablen `antwort` weder das Zeichen `'j'` noch das Zeichen `'J'` enthält:
(für `antwort = 'A'`)
 7. ob der Wert der `int`-Variablen `zaehler` nicht im Intervall `[5,25]` liegt:
(für `zaehler = 30`)
-

5.5 Bit-Operatoren

5.5.1 Überprüfungen mit Bit-Operatoren *Programmierung(4;<10)*

Erstellen Sie ein Programm, das die Werte der Ausdrücke der unten angegebenen Überprüfungen einer int-Variable `wert` zuweist und diese dann als Dezimalzahl, also mit

```
printf("n.Wert: %d\n", wert);
```

ausgibt.

Bei den Überprüfungen sollten Sie nur logische Operatoren und Bit-Operatoren benutzen. So könnte z.B. mit dem Ausdruck

```
x & 0x8000
```

überprüft werden, ob der Wert einer **short**-Variablen negativ ist, da bei dieser Verknüpfung alle Bits außer dem ersten auf 0 gesetzt werden. Ist das 1.Bit 1 (Negative Zahl), so ist der Wert dieser Verknüpfung verschieden von 0, was in C als TRUE gewertet wird. Ist dagegen das 1.Bit 0 (positive Zahl), so ist der Wert dieser Verknüpfung 0, was in C als FALSE gewertet wird. Geben Sie nun entsprechende Ausdrücke an, die überprüfen,

-
1. ob der Wert der short-Variablen `x` ungerade ist.
(für `x = 7`)
 2. ob der Wert der unsigned short-Variablen `u` grösser als 255 ist.
(für `u = 256`)
 3. ob das 7.Bit (von links her gezählt) in der unsigned short-Variablen `u` auf 1 gesetzt ist.
(für `u = 520`)
 4. ob der Wert der short-Variablen `x` im Intervall `[0,127]` liegt.
(für `x = 120`)
 5. ob der Wert der unsigned short-Variablen `u` durch 4 teilbar ist.
(für `u = 43`)
-

5.6 Shift-Operatoren

5.6.1 Vertauschen von zwei Bytes *Programmierung(4;<10)*

In einer **unsigned short**-Variable `x` sei der Wert **0x12ab** gespeichert. Mit einer Zuweisung an `x` sollen nun die beiden Bytes dieser Variable `x` vertauscht werden. Nach dieser Zuweisung sollte in `x` der Wert **0xab12** stehen.

Erstellen Sie nun ein Programm, das den ursprünglichen und den neuen Wert als Hex-Zahl ausgibt. Das Programm muss auch für jeden anderen zugewiesenen Wert den neuen Wert mit den vertauschten Bytes ausgeben.

Hinweis: Schieben Sie die einzelnen Bytes jeweils an die neue Position und setzen Sie dann die beiden geschobenen Bytes mit dem binären Oder-Operator wieder zu einem neuen 2-Byte Wert zusammen.

```
x = .....
```

5.6.2 Duale Ausgabe eines Bytes *Programmierung(3;<20)*

Schreiben Sie ein C-Programm *bytedual.c*, das Ihnen das Bitmuster des Wertes in der **char**-Variablen `zeichen` ausgibt. Es bleibt dabei Ihnen überlassen, welche **char**-Konstante Sie zuvor der Variablen `zeichen` zuweisen. Für z. B. `zeichen = 'e'` sollte das Programm das zugehörige Bitmuster (ASCII-Code)

```
01100101
```

ausgeben. Das Programm muss auch für jede andere Zuweisung das korrekte Bitmuster ausgeben.

Hierzu noch ein Hinweis:

Geben Sie jedes einzelne Bit mit einem `printf()` aus. Sie müssen also `printf()` achtmal aufrufen, um jedes einzelne Bit von `zeichen` auszugeben.

Die Idee dieses Programms ist, das auszugebende Bit mit `»` an die letzte Stelle zu schieben und dann eventuell zuvor stehende Einsen zu löschen. Das Löschen der vorderen 7 Bits wird dadurch erreicht, dass das mit `»` entstandene Bitmuster und die Zahl 1 mit dem Operator `&` verknüpft werden. Um z. B. das dritte Bit von vorne auszugeben, wird

```
printf("%d", (zeichen » 5) & 1)
```

aufgerufen:

```
zeichen:      01100101 = 0xf
zeichen >> 5: 00000011 = 3
              00000001 = 1
-----
(zeichen >> 5) & 1: 00000001 = 1 (Ausgabe durch printf())
```

Kapitel 6

Symbolische Konstanten

6.1 Konstanten-Definition mit #define

6.1.1 Volumen und Oberfläche einer Kugel *Mathematik(3;<20)*

Erstellen Sie ein C-Programm *kugel.c*, das mit `scanf()` den Radius einer Kugel einliest und dann den Umfang ($2\pi r$), das Volumen ($\frac{4}{3}r^3\pi$) und die Oberfläche ($4\pi r^2$) dieser Kugel ausgibt. In *kugel.c* sollten Sie dabei Konstanten für π und 4π definieren. Nachfolgend sind Beispiele für mögliche Programmabläufe gegeben.

```
Radius der Kugel : 3 ↵
Umfang=18.849556
Volumen=113.097336
Oberflaeche=113.097336
```

```
Radius der Kugel : 34.5 ↵
Umfang=216.769897
Volumen=172006.906250
Oberflaeche=14957.123047
```

6.1.2 Lichtjahre in Kilometer umrechnen *Physik(3;<20)*

Erstellen Sie ein C-Programm *ljahrkm.c*, das bei der Eingabe von Lichtjahren die diesen Lichtjahren entsprechenden Kilometer ausgibt; 1 Lichtjahr = 9,4605 Billionen km.

Mögliche Programmabläufe:

```
Wieviele Lichtjahre: 3.4 ↵
-----> 32165700042752.000000 Kilometer
```

```
Wieviele Lichtjahre: 1000000 ↵
-----> 9460500411082342400.000000 Kilometer
```

6.1.3 Das Phänomen der entfesselten Erde Allgemein(4;<20)

Würde man um die Erde ein Seil legen, dann hätte dieses Seil eine bestimmte Länge. Würde man nun aber dieses Seil aufschneiden und einen Meter neues Seil einfügen, um wie viele Zentimeter würde dann das verlängerte Seil (als neuer Kreis um die Erde gelegt) von der Erde abstehen? Erstellen Sie ein C-Programm *erdumf.c*, das zunächst den Radius für einen Körper (wie z.B. die Erde) einliest, und dann den Abstand (in cm) ausgibt, den das um diesen Körper gelegte Seil bei einer Verlängerung um einen Meter hätte. In diesem Programm sollten Sie mit **double**-Variablen arbeiten. Einlesen in **double**-Variablen erfolgt mit der Format-Angabe `%lf`. Für die Ausgabe von **double**-Variablen muss ebenfalls `%lf` verwendet werden, wie z.B:

```
scanf("%lf", &radius);
printf("%.2lf", abstand); /* double-Var. mit 2 Nachkommastellen ausgeben*/
```

Beispiel für einen möglichen Programmablauf:

```
Welchen Radius hat Körper, um den Seil gelegt wird (in m) ? 6378388 (↩)

Nach Verlaengerung des Seils um 1 Meter
steht es um 15.92 cm ab
```

6.2 Konstanten-Definition mit const

6.2.1 Fallzeit für einen Körper Physik(3;<20)

Erstellen Sie ein C-Programm *fallzeit.c*, das die Zeit berechnet, die ein Körper wie z.B. ein Stein für den Fall aus einer bestimmten Höhe benötigt. Die Höhe ist dabei mit `scanf()` einzulesen. Bei diesem Programm sollten Sie die Gravitations-Konstante g ($g = 9.80665 \frac{m}{s^2}$) mit **const** definieren. Die Zeit für den Fall eines Körpers lässt sich nach folgender Formel berechnen:

$$t = \sqrt{\frac{2h}{g}}$$

Das zur Wurzelberechnung benötigte Programmteil hat den Namen `sqrt()`. Damit Sie `sqrt()` verwenden können, sollten Sie in Ihrem Programm *fallzeit.c* als erstes die folgende Zeile angeben:

```
#include <math.h>
```

und das Kompilieren und Linken des Programms mit folgender Kommandozeile durchführen:

```
cc -o fallzeit fallzeit.c -lm
```

Beispiel für mögliche Programmabläufe:

```
Hoehe des Koerpers (in Meter): 100 (↩)
--->Fallzeit: 4.516007 Sek.
```

```
Hoehe des Koerpers (in Meter): 1000 (↩)
---> Fallzeit: 14.280869 Sek.
```

6.2.2 Benzinverbrauch und Geschwindigkeit *Allgemein(3;<50)*

Erstellen Sie ein C-Programm *benzinv.c*, das die durchschnittliche Geschwindigkeit und den durchschnittlichen Benzinverbrauch für eine Autofahrt berechnet. Die gefahrene Zeit (Stunden und Minuten), die gefahrenen Kilometer und die gebrauchten Liter sind dazu einzugeben, wie z. B.:

```
Gefahrene Stunden: 2
Gefahrene Minuten: 7
Gefahrene Kilometer: 234
Gebrauchte Liter: 18.9

Durchschnittl. Geschwindigkeit: 110.55 km/h (30.71 m/s)
Durchschnittl. Benzinverbrauch: 8.08 l/100 km
```

```
Gefahrene Stunden: 9
Gefahrene Minuten: 43
Gefahrene Kilometer: 782
Gebrauchte Liter: 65.7

Durchschnittl. Geschwindigkeit: 80.48 km/h (22.36 m/s)
Durchschnittl. Benzinverbrauch: 8.40 l/100 km
```

6.2.3 Zinsertrag für ein bestimmtes Kapital *Wirtschaft(3;<20)*

Für ein Anfangskapital K_0 sind in Abhängigkeit vom Festlegungszeitraum T (Tage) und dem Jahreszinsfuß (Z in %) die Zinsen und das Endkapital zu errechnen. Dazu benutzt man folgende Formeln:

$$\text{Zinsen} = \frac{K_0 * Z * T}{100 * 360} \qquad \text{Endkapital} = K_0 + \text{Zinsen}^1$$

Erstellen Sie ein C-Programm *zinskapi.c*, das zunächst ein Anfangskapital, den dafür gewährten Zinsfuß (in Prozent) und die Anlegedauer einliest, und dann die dafür zu zahlenden Zinsen und das resultierende Endkapital ausgibt, wie z. B.:

```
Anfangskapital: 120000
Jahreszins (in %): 7.43
Wieviele Tage soll Kapital angelegt werden: 430

Anfangskapital:           120000.00
Zinsen (nach 430 Tage):   10649.67
-----
Endkapital:               130649.66
```

```
Anfangskapital: 13500
Jahreszins (in %): 5.7
Wieviele Tage soll Kapital angelegt werden: 200

Anfangskapital:           13500.00
Zinsen (nach 200 Tage):   427.50
-----
Endkapital:               13927.50
```

¹Bei der Zinsberechnung werden immer 360 Tage für ein Jahr angenommen

Kapitel 7

Ein- und Ausgabe

7.1 Headerdateien und #include

7.1.1 Umrechnung von Geschwindigkeiten *Physik(3;<50)*

Erstellen Sie ein C-Programm *vumrech.c*, das eine zurückgelegte Strecke (in Meter) und die dafür benötigte Zeit (in Sekunden) einliest, bevor es dann die Geschwindigkeit in m/s, km/h, km/tag und m/tag ausgibt. Die Umrechnungsfaktoren sollten Sie dabei in der eigenen Headerdatei *faktor.h* definieren:

```
#define MS_NACH_KMH      .....  
#define MS_NACH_KMTAG   .....  
#define MS_NACH_MTAG     .....
```

Diese Umrechnungsfaktoren sollten Sie dann in Ihrem Programm *vumrech.c* verwenden. Um dies tun zu können, müssen Sie dort aber Folgendes angeben:

```
#include "faktor.h"
```

Mögliche Abläufe dieses Programms *vumrech.c*:

```
Gib Strecke ein (in Meter): 1 (↵)  
Gib Zeit ein, die dafuer benoetigt wird (in Sekunden): 1 (↵)  
Dies entspricht folgender Geschwindigkeit:  
1.000000 m/sec =  
3.600000 km/h =  
86400.000000 m/Tag =  
86.400002 km/Tag
```

```
Gib Strecke ein (in Meter): 123 (↵)  
Gib Zeit ein, die dafuer benoetigt wird (in Sekunden): 3.5 (↵)  
Dies entspricht folgender Geschwindigkeit:  
35.142857 m/sec =  
126.514282 km/h =  
3036342.750000 m/Tag =  
3036.342773 km/Tag
```

7.2 Ein- und Ausgabe eines Zeichens

7.2.1 Quersumme zu einer 5-stelligen Zahl *Mathematik(4;<20)*

Erstellen Sie ein C-Programm *quersum.c*, das eine 5-stellige Zahl Zeichen für Zeichen einliest, also nicht als numerischen Wert, und dann die Quersumme zu dieser Zahl ausgibt. Ein Beispiel für einen möglichen Ablauf dieses Programms könnte sein:

```
Gib 5-stellige Zahl ein: 98934 ↵
Die Quersumme dieser Zahl ist 33
```

7.3 Die Ausgabe mit printf()

7.3.1 Ausgeben eines Rahmens *Allgemein(6;<20)*

Erstellen Sie ein C-Programm *rahmen.c*, das einen Rahmen am Bildschirm ausgibt. Der Rahmen soll immer 5 Zeilen lang sein. Wie breit er sein soll, soll der Benutzer eingeben.

Hinweis für alle, die schon C-Vorkenntnisse haben:

Das Programm muss ohne Schleifen gelöst werden.

```
Wie breit soll Rahmen werden: 5 ↵
+-----+
|       |
|       |
|       |
+-----+
```

```
Wie breit soll Rahmen werden: 45 ↵
+-----+
|                                     |
|                                     |
|                                     |
+-----+
```

```
Wie breit soll Rahmen werden: 75 ↵
+-----+
|                                     |
|                                     |
|                                     |
+-----+
```

7.4 Die Eingabe mit scanf()

7.4.1 Dezimal- und Hexawert zu einer Oktalzahl *C-Syntax(2;<10)*

Erstellen Sie ein C-Programm *okdezhex.c*, das eine Oktalzahl einliest und dann die dieser Oktalzahl entsprechende Dezimal- und Hexadezimalzahl ausgibt. Zur Erkennung einer falschen Eingabe ist der Rückgabewert von `scanf()` nach dem Einlesen auszugeben.

```
Gib eine Oktalzahl ein: 7654321 ↵
---> 7654321(8) = 2054353(10) = 1f58d1(16)
```

Kapitel 8

Datentypumwandlungen

8.1 Implizite Datentypumwandlungen

8.1.1 Reißleine von Fallschirm wann ziehen? *Physik(3;<20)*

Die Höhe, bei der ein Fallschirmspringer spätestens seine Reißleine ziehen muss, um unversehrte zu landen, beträgt etwa 650 Meter. Erstellen Sie ein C-Programm *falschir.c*, das die Absprunghöhe als Ganzzahl einliest und dann ausgibt, wie viele Sekunden (als Ganzzahl) dem Fallschirmspringer bleiben, bis er die Reißleine ziehen muss.

Mögliche Abläufe dieses Programms *falschir.c*:

```
In welcher Hoehe verlaesst der Fallschirmspringer den Flieger: 3400 ↵  
---> Springer muss nach 23 Sekunden die Reissleine ziehen
```

```
In welcher Hoehe verlaesst der Fallschirmspringer den Flieger: 45000 ↵  
---> Springer muss nach 95 Sekunden die Reissleine ziehen
```

8.2 Explizite Datentypumwandlungen

8.2.1 Prozentzahlen für Kandidaten bei einer Wahl *Allgemein(3;<30)*

Bei einer Wahl stellen sich vier Kandidaten zur Verfügung. Erstellen Sie ein C-Programm *kandproz.c*, das zunächst für jeden einzelnen der Kandidaten einliest, wie viele Stimmen er erhalten hat. Die erhaltenen Stimmen sollen dabei in **int**-Variablen eingelesen werden. Danach soll dieses C-Programm die Prozentzahlen für jeden einzelnen dieser Kandidaten ausgeben. Möglicher Ablauf dieses Programms *kandproz.c*:

```
Stimmen für den 1. Kandidaten: 129089 ↵
Stimmen für den 2. Kandidaten: 200234 ↵
Stimmen für den 3. Kandidaten: 89012 ↵
Stimmen für den 4. Kandidaten: 199456 ↵
1. Kandidat: 20.90%
2. Kandidat: 32.41%
3. Kandidat: 14.41%
4. Kandidat: 32.29%
```

Kapitel 11

Die if-Anweisung

11.1 Schaltjahre *C-Syntax(3;<30)*

Erstellen Sie ein C-Programm *schalt.c*, das eine Jahreszahl einliest und dann ausgibt, ob es sich bei diesem Jahr um ein Schaltjahr handelt oder nicht. Das in Abbildung 11.1 gezeigte Struktogramm zeigt die Regeln für ein Schaltjahr. Setzen Sie nun dieses Struktogramm in das C-Programm *schalt.c* um!

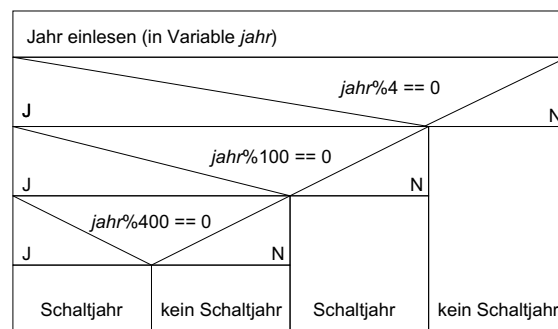


Abbildung 11.1: Struktogramm zur Bestimmung eines Schaltjahres

Beispiele für den möglichen Ablauf des Programms *schalt.c*:

```
Gib ein Jahr ein: 1900 (↩)
---> kein Schaltjahr
```

```
Gib ein Jahr ein: 2000 (↩)
---> Schaltjahr
```

11.2 Tageszeit abhängig grüssen *Allgemein(3;<30)*

Erstellen Sie ein C-Programm *gruss.c*, welches eine Stunde einliest, und abhängig von der Stunde folgendes ausgibt.

Stunde	Ausgabe
23,0,1,2,3,4,5	Gute Nacht
6,7,8,9,10	Guten Morgen
11,12,13	Mahlzeit
14,15,16,17	Schönen Nachmittag
18,19,20,21,22	Guten Abend
sonst	keine erlaubte Stunden-Angabe

```
Gib die Stunde der momentanen Uhrzeit ein: 1 (↩)
---> Gute Nacht
```

```
Gib die Stunde der momentanen Uhrzeit ein: 9 (↩)
---> Guten Morgen
```

```
Gib die Stunde der momentanen Uhrzeit ein: 12 (↩)
---> Mahlzeit
```

```
Gib die Stunde der momentanen Uhrzeit ein: 16 (↩)
---> Schönen Nachmittag
```

```
Gib die Stunde der momentanen Uhrzeit ein: 21 (↩)
---> Guten Abend
```

```
Gib die Stunde der momentanen Uhrzeit ein: 24 (↩)
---> 24 ist keine erlaubte Stunden-Angabe
```

11.3 Zahl x durch Zahl y teilbar? *Mathematik(2;<30)*

Erstellen Sie ein C-Programm *teilbar.c*, das zwei ganze Zahlen x und y einliest und dann ausgibt, ob x durch y teilbar ist.

Mögliche Abläufe dieses Programms *teilbar.c*:

```
Zahl x durch Zahl y teilbar ?
=====
Dieses Programm sagt Ihnen, ob eine Zahl x durch eine Zahl y
teilbar ist. Dazu muessen nur die beiden ganzen Zahlen x und y eingeben.
x? 1000 (↩)
y? 8 (↩)
    1000 ist durch 8 .....teilbar
```

```
....
x? 721 (↩)
y? 3 (↩)
    721 ist durch 3 .....nicht teilbar
```

Kapitel 13

Die switch-Anweisung

13.1 Menü-Auswahl *Programmierung(2;<40)*

Erstellen Sie ein C-Programm *menue.c*, das eine Menü-Auswahl einliest, und dem Benutzer unter Verwendung von switch dann seine getroffene Wahl ausgibt.

Mögliche Abläufe des Programms *menue.c*:


```
Hauptmenue
=====

(A) endern
(B) eenden
(D) rucken
(E) ingeben
(L) oeschen

Was wuenschen Sie zu tun ? d 
..... Sie haben (D)rucken gewaehlt .....
```

```
Hauptmenue
=====

(A) endern
(B) eenden
(D) rucken
(E) ingeben
(L) oeschen

Was wuenschen Sie zu tun ? x 
..... Ihre Wahl 'X' ist unerlaubt .....
```


13.2 Flächen zu geometrischen Figuren *Mathematik(3;<60)*

Erstellen Sie ein C-Programm *flaeche.c*, bei dem der Benutzer wählen kann, welche der folgenden Flächen er sich berechnen lassen möchte:

Quadrat: $A = a \cdot a$ (a ist die einzugebende Seitenlänge)

Rechteck: $A = a \cdot b$ (a und b sind die einzugebenden Seitenlängen)

Kreis: $A = a \cdot a \cdot \pi$ (a ist der Radius des Kreises)

Ellipse: $A = a \cdot b \cdot \pi$ (a ist Radius der x- und b Radius der y-Achse der Ellipse)

An dieser Formel für die Ellipse ist zu erkennen, dass eine Ellipse zu einem Kreis degeneriert, wenn a und b gleich sind.

Mögliche Abläufe dieses Programms *flaeche.c*:

```

Quadrat:    q
Rechteck:   r
Kreis:      k
Ellipse:    e
Deine Wahl? q
Seitenlaenge des Quadrats: 21.4
..... Flaechе = 457.96

```

```

Quadrat:    q
Rechteck:   r
Kreis:      k
Ellipse:    e
Deine Wahl? e
Radius der x-Achse: 12.3
Radius der y-Achse: 5.6
..... Flaechе = 216.39

```

```

Quadrat:    q
Rechteck:   r
Kreis:      k
Ellipse:    e
Deine Wahl? r
Laenge der 1. Rechteckseite: 7
Laenge der 2. Rechteckseite: 8
..... Flaechе = 56.00

```

```

Quadrat:    q
Rechteck:   r
Kreis:      k
Ellipse:    e
Deine Wahl? x
..... 'x' ist keine erlaubte Wahl

```

13.3 Folgedatum zu einem Datum Allgemein(4;<60)

Erstellen Sie ein C-Programm *naechtag.c*, das ein Datum einliest, und dann das Datum des darauffolgenden Tages ausgibt. Schaltjahre haben im Februar immer 29 (anstelle von 28) Tagen. Ein Schaltjahr liegt immer dann vor, wenn die Jahreszahl durch 4, aber nicht durch 100, oder aber, wenn die Jahreszahl durch 400 teilbar ist. Mögliche Abläufe dieses C-Programms *naechtag.c* sind z.B.:

```
Folgedatum zu einem Datum
=====

Dieses Programm liest ein Kalender-Datum ein, und gibt dann
das Datum des nachfolgenden Tages aus.
```

```
Gib Datum ein (tt.mm.jjjj): 28.2.1997 ⏪
28.2.1997 ---> 1.3.1997
```

```
.....
Gib Datum ein (tt.mm.jjjj): 29.2.2000 ⏪
29.2.2000 ---> 1.3.2000
```

```
Gib Datum ein (tt.mm.jjjj): 29.2.1999 ⏪
29.2.1999 ist kein gueltiges Datum
```

```
Gib Datum ein (tt.mm.jjjj): 28.2.1992 ⏪
28.2.1992 ---> 29.2.1992
```

```
Gib Datum ein (tt.mm.jjjj): 28.2.1800 ⏪
28.2.1800 ---> 1.3.1800
```

```
Gib Datum ein (tt.mm.jjjj): 31.9.1778 ⏪
31.9.1778 ist kein gueltiges Datum
```

```
Gib Datum ein (tt.mm.jjjj): 31.12.1985 ⏪
31.12.1985 ---> 1.1.1986
```

```
Gib Datum ein (tt.mm.jjjj): 30.6.2068 ⏪
30.6.2068 ---> 1.7.2068
```


13.4 Fläche, Umfang und Radius eines Kreises *Mathematik(3;<60)*


Erstellen Sie ein C-Programm *kreis.c*, bei dem zunächst der Benutzer wählen kann, ob er den Umfang, die Fläche oder den Radius eines Kreises eingeben möchte. Nach dieser Wahl muss er dann die betreffende Größe eingeben, bevor das Programm die beiden fehlenden Größen berechnet und ausgibt.

Mögliche Abläufe dieses Programms *kreis.c*:

```
Flaeche, Umfang und Radius eines Kreises
=====
```


```
Du kannst waehlen, was du eingeben moechtest.
Ich berechne Dir dann die 2 fehlenden Groessen.
    Flaeche eingeben:    f
    Umfang eingeben:    u
    Radius eingeben:    r
```


```
Deine Wahl? r 
```

```
Radius? 12.3 
```

```
..... Radius   = 12.30
..... Flaeche   = 475.29
..... Umfang    = 77.28
```


```
.....
    Flaeche eingeben:    f
    Umfang eingeben:    u
    Radius eingeben:    r
```

```
Deine Wahl? u 
```

```
Umfang? 246.6 
```

```
..... Radius   = 39.25
..... Flaeche   = 4839.23
..... Umfang    = 246.60
```

```
.....
    Flaeche eingeben:    f
    Umfang eingeben:    u
    Radius eingeben:    r
```

```
Deine Wahl? k 
```

```
.....'k' ist keine erlaubte Wahl
```

Kapitel 15

Die for-Anweisung

15.1 Berechnung der harmonischen Reihe *Mathematik(2;<20)*

Erstellen Sie ein C-Programm *harmon.c*, das die harmonische Reihe berechnet, wobei der Endwert *n* dabei einzugeben ist:

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \dots + \frac{1}{n}$$

Mögliche Abläufe dieses Programms *harmon.c*:

```
Bis zu welchem n soll diese Reihe berechnet werden: 3 (↩)
Summe bis 1/3: 1.8333333
```

```
Bis zu welchem n soll diese Reihe berechnet werden: 10 (↩)
Summe bis 1/10: 2.9289683
```

```
Bis zu welchem n soll diese Reihe berechnet werden: 1000 (↩)
Summe bis 1/1000: 7.4854709
```

```
Bis zu welchem n soll diese Reihe berechnet werden: 1000000 (↩)
Summe bis 1/1000000: 14.3927267
```

15.2 Summe von ungeraden Zahlen *Mathematik(2;<20)*

Erstellen Sie ein C-Programm *sumunger.c*, das bei 1 beginnend eine Summe von ungeraden Zahlen berechnet. Bis zu welchem Endwert dabei die ungeraden Zahlen aufzuaddieren sind, ist einzugeben. Mögliche Abläufe dieses Programms *sumunger.c*:

```
Bis zu welchem n sollen alle ungeraden Zahlen aufaddiert werden: 5 (↩)
Summe der ungeraden Zahlen bis 5: 9
```

```
Bis zu welchem n sollen alle ungeraden Zahlen aufaddiert werden: 100 (↩)
Summe der ungeraden Zahlen bis 100: 2500
```

```
Bis zu welchem n sollen alle ungeraden Zahlen aufaddiert werden: 3600 (↩)
Summe der ungeraden Zahlen bis 3600: 3240000
```

15.3 Berechnung der Leibniz-Reihe *Mathematik(3;<20)*

Erstellen Sie ein C-Programm *leibniz.c*, das die Leibniz-Reihe berechnet:

$$\frac{\pi}{4} = \frac{1}{1} - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} \mp \dots$$

Wie viele Brüche zu addieren bzw. zu subtrahieren sind, ist einzugeben.
Mögliche Abläufe dieses Programms *leibniz.c*:

```
Wieviele Brueche sollen addiert bzw. subtrahiert werden: 3
Summe bis zum 3.Glied: 0.8666667 ==> PI=3.4666667
```

```
.....
Wieviele Brueche sollen addiert bzw. subtrahiert werden: 100
Summe bis zum 100.Glied: 0.7828982 ==> PI=3.1315929
```

```
Wieviele Brueche sollen addiert bzw. subtrahiert werden: 1000
Summe bis zum 1000.Glied: 0.7851482 ==> PI=3.1405927
```

```
Wieviele Brueche sollen addiert bzw. subtrahiert werden: 100000
Summe bis zum 100000.Glied: 0.7853957 ==> PI=3.1415827
```

15.4 Berechnung der Exponential-Reihe *Mathematik(4;<30)*

Erstellen Sie ein C-Programm *exporeih.c*, das die Exponential-Reihe berechnet:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \dots + \frac{x^n}{n!}$$

Die Potenz *x* und der Endwert *n* ist dabei einzugeben.
Mögliche Abläufe dieses Programms *exporeih.c*:

```
x ? 1
n ? 3
e hoch 1 = 2.6666667 (bis zum 3.Glied)
```

```
x ? 1
n ? 20
e hoch 1 = 2.7182818 (bis zum 20.Glied)
```

```
x ? 3
n ? 10
e hoch 3 = 20.0796652 (bis zum 10.Glied)
```

```
x ? 3
n ? 20
e hoch 3 = 20.0855369 (bis zum 20.Glied)
```

15.5 Anzahl der Handschläge auf einer Party *Allgemein(3;<20)*

Dies ist die altbekannte Geschichte vom Händeschütteln. n Personen treffen sich auf einer Party. Jeder schüttelt dabei jedem die Hände. Wieoft werden insgesamt Hände geschüttelt. Wenn z.B. 10 Personen auf der Party sind, so schüttelt die erste Person 9 anderen Personen die Hand, die zweite noch 8 anderen, die dritte 7 anderen usw.

Erstellen Sie ein C-Programm *party.c*, das einliest, wie viele Personen auf der Party sind, und dann ausgibt, wieoft Hände auf dieser Party geschüttelt werden.

Mögliche Abläufe dieses Programms *party.c*:

```
Haendeschuetteln aller Personen auf einer Party
=====
```

```
Wieviele Personen sind auf der Party: 10 (↵)
Es werden 45 mal die Haende geschuettelt
```

```
.....
Wieviele Personen sind auf der Party: 100 (↵)
Es werden 4950 mal die Haende geschuettelt
```

```
.....
Wieviele Personen sind auf der Party: 500 (↵)
Es werden 124750 mal die Haende geschuettelt
```

15.6 Alle Teiler zu einer Zahl *Mathematik(4;<30)*

Erstellen Sie ein C-Programm *teiler.c*, das alle Teiler zu einer Zahl, die einzugeben ist, ermittelt und ausgibt. 1 und die Zahl selbst sollen dabei nicht mit ausgegeben werden.

```
Mögliche Abläufe dieses Programms teiler.c:
Alle Teiler zu einer Zahl bestimmen
```

```
=====
```

```
Gib eine Zahl ein: 100 (↵)
100 ist teilbar durch: 2, 4, 5, 10, 20, 25, 50
```

```
.....
Gib eine Zahl ein: 17 (↵)
17 ist eine Primzahl
```


Kapitel 16

Die while-Anweisung

Hinweis: Ab diesem Kapitel sind alle Benutzereingaben immer auf im Programmkontext sinnvolle Eingaben (d. h. i.A. auf korrekte Rückgabewerte von `scanf()` mithilfe einer Schleife) zu überprüfen.

16.1 Fritz und Hans essen Äpfel *Allgemein(3;<40)*

Fritz und Hans haben gemeinsam x Äpfel gekauft. In der Zeit, in der Fritz 5 Äpfel ißt, ißt Hans 3. Wie viele Äpfel hat jeder gegessen, wenn keine mehr übrig sind. Erstellen Sie ein C-Programm *aepfel.c*, das zunächst einliest, wie viele Äpfel Fritz und Hans zusammen gekauft haben, und dann die in jeder Runde von Fritz bzw. Hans gegessenen Äpfel einschließlich der noch übrigen Äpfel in Form einer Tabelle ausgibt, wie z. B.:

```
Wieviele Aepfel haben Fritz und Hans gekauft: 114 (↩)
Sorry, aber die Zahl der Aepfel muss durch 8 teilbar sein
Gib eine neue Zahl ein: 122 (↩)
Sorry, aber die Zahl der Aepfel muss durch 8 teilbar sein
Gib eine neue Zahl ein: 120 (↩)
```

Runde	Fritz	Hans	Rest
1	5	3	112
2	10	6	104
3	15	9	96
4	20	12	88
5	25	15	80
6	30	18	72
7	35	21	64
8	40	24	56
9	45	27	48
10	50	30	40
11	55	33	32
12	60	36	24
13	65	39	16
14	70	42	8
15	75	45	0

16.2 Eingabe einer geraden Zahl zwischen 1 und 100 *Allgemein(3;<20)*

Erstellen Sie ein Programm, das eine gerade Zahl zwischen 1 und 100 einliest, die Eingabe überprüft und bei falschen Eingaben zur Wiederholung auffordert. Nach korrekter Eingabe ist die entsprechende Zahl auszugeben.

16.3 Verschlüsseln mit Verschiebchiffre *Allgemein(3;<30)*

Es soll hier ein C-Programm *vciffre.c* erstellt werden, welches Nachrichten verschlüsselt. Als Chiffrieralgorithmus soll dabei ein einfacher Verschiebchiffre verwendet werden. Der einzugebende Schlüssel x legt fest, dass für jeden Kleinbuchstaben aus dem Originaltext dessen x . Nachfolger unter den Kleinbuchstaben und für jeden Großbuchstaben dessen x . Nachfolger unter den Großbuchstaben auszugeben ist. Das entsprechende Alphabet denkt man sich dabei, wie in Abbildung 16.1 gezeigt, zyklisch fortgesetzt.

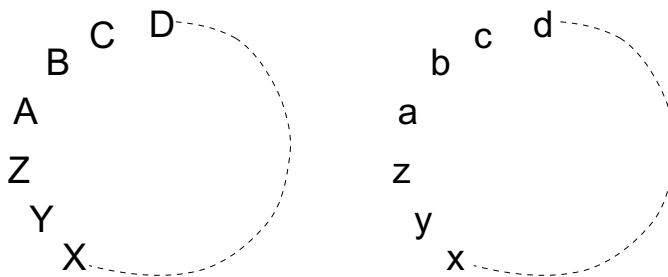


Abbildung 16.1: Alphabet zyklisch fortgesetzt

Tritt im Eingabetext ein Zeichen auf, das nicht im englischen Alphabet enthalten ist, wie z.B. ä oder ?, so soll es unverschlüsselt wieder ausgegeben werden.

Möglicher Ablauf dieses Programms *vciffre.c*:

```
Verschlüsseln einer Nachricht mit Verschiebchiffre
=====

Der wievielte Nachfolger soll immer genommen werden ? 3 (↩)

Gib nun den zu verschlüsselten Text ein:
In diesem Beispiel wird ein (↩)
Lq glhvhp Ehlvslho zlug hlq
einfacher Verschiebchiffre gezeigt. (↩)
hlqidfkhu Yhuvfklhehfkliiuh jhchljw.
(Strg)(D)
```

16.4 Minimum und Maximum von n Zahlen Allgemein(4;<50)

Erstellen Sie ein C-Programm *minmax.c*, das n Zahlen einliest und dann die kleinste und die größte der eingegebenen Zahlen wieder ausgibt.

```
Wie viele Zahlen: 5 (↵)
1.Zahl: 7 (↵)
2.Zahl: 3.5 (↵)
3.Zahl: 14.32 (↵)
4.Zahl: 3.25 (↵)
5.Zahl: 74 (↵)
--> Die 5.Zahl (74.00) war die groesste Zahl
--> Die 4.Zahl (3.25) war die kleinste Zahl
```

```
Wie viele Zahlen: 7 (↵)
1.Zahl: 128.43 (↵)
2.Zahl: 1200 (↵)
3.Zahl: -45 (↵)
4.Zahl: 1320 (↵)
5.Zahl: 54 (↵)
6.Zahl: 12 (↵)
7.Zahl: -7 (↵)
--> Die 4.Zahl (1320.00) war die groesste Zahl
--> Die 3.Zahl (-45.00) war die kleinste Zahl
```

16.5 Fibonacci-Zahlen Mathematik(4;<30)

Erstellen Sie ein C-Programm *fibonacci.c*, das die Fibonacci-Zahlen am Bildschirm ausgibt. Die beiden ersten Zahlen der Fibonacci-Zahlenfolge sind 1 und die weiteren Zahlen ergeben sich immer aus der Summe der beiden vorhergehenden Zahlen in der Folge. Der Endwert der auszugebenden Zahlenfolge ist einzugeben. Bei der Ausgabe der Zahlenfolge soll die Ausgabe immer nach 20 Zahlen angehalten werden, um dem Benutzer die ausgegebenen Zahlen lesen zu lassen. Erst, wenn der Benutzer die (↵)-Taste drückt, werden die nächsten 20 Fibonacci-Zahlen ausgegeben.

Mögliche Abläufe dieses Programms *fibonacci.c*:

```
Fibonacci-Zahlen
=====

Bis wohin ? 2000 (↵)
1.          1
2.          1
3.          2
4.          3
5.          5
6.          8
7.         13
8.         21
9.         34
10.        55
11.        89
12.       144
13.       233
14.       377
15.       610
16.      987
17.     1597
```

```
Fibonacci-Zahlen
=====

Bis wohin ? 1234567890 (←)

1.      1
2.      1
3.      2
4.      3
5.      5
6.      8
7.     13
8.     21
9.     34
10.    55
11.    89
12.   144
13.   233
14.   377
15.   610
16.   987
17.  1597
18.  2584
19.  4181
20.  6765

Weiter mit Return..... (←)

21.    10946
22.    17711
23.    28657
24.    46368
25.    75025
26.   121393
27.   196418
28.   317811
29.   514229
30.   832040
31.  1346269
32.  2178309
33.  3524578
34.  5702887
35.  9227465
36. 14930352
37. 24157817
38. 39088169
39. 63245986
40. 102334155

Weiter mit Return..... (←)

41. 165580141
42. 267914296
43. 433494437
44. 701408733
45. 1134903170
```

16.6 Quersumme einer Zahl *Mathematik(4;<20)*

Erstellen Sie ein C-Programm *quersum.c*, das eine ganze Zahl einliest und dann deren Quersumme ausgibt. Die Quersumme einer Zahl ist die Summe aller ihrer Ziffern.

Mögliche Abläufe dieses Programms *quersum.c*:

```
Quersumme zu einer Zahl
=====
Zahl ? 42675 (↩)
Quersumme zu 42675 ist: 24
```

```
.....
Zahl ? 67894356 (↩)
Quersumme zu 67894356 ist: 48
```

16.7 Duale BCD-Darstellung einer Zahl *Mathematik(4;<50)*

Erstellen Sie ein C-Programm *dualbcd.c*, das eine natürliche Zahl einliest, und dann deren Dualdarstellung und deren BCD-Darstellung am Bildschirm ausgibt. Bei der BCD-Darstellung wird jede Ziffer einzeln in 4 Bits codiert, wie z.B. 2365

```
0010 0011 0110 0101
  2    3    6    5
```

Für die reine Dualdarstellung (nicht BCD) sollten Sie kein Konvertierungs-Verfahren verwenden, sondern unter Verwendung von Bitoperatoren auf die bereits im Rechner vorliegende Dualdarstellung der Zahl zugreifen.

Mögliche Abläufe dieses Programms *dualbcd.c*:

```
Dual- und duale BCD-Darstellung fuer eine Zahl
=====
Zahl: 12345678 (↩)

Dualdarstellung zu 12345678: 00000000 10111100 01100001 01001110
Duale BCD-Darstellung zu 12345678: 0001 0010 0011 0100 0101 0110 0111 1000
```

```
Dual- und duale BCD-Darstellung fuer eine Zahl
=====
Zahl: -345 (↩)

Zahl muss im Intervall [0,99999999] liegen

Neue Zahl: 2365 (↩)

Dualdarstellung zu 2365: 00000000 00000000 00001001 00111101
Duale BCD-Darstellung zu 2365: 0000 0000 0000 0000 0010 0011 0110 0101
```


Kapitel 17

Die do...while-Anweisung

17.1 Zahlen raten *Allgemein(3;<50)*

Erstellen Sie ein C-Programm *zahlrat.c*, das sich zufällig eine Zahl aus dem Intervall $[1,x]$ denkt. x ist dabei vom Benutzer einzugeben. Danach soll der Benutzer versuchen, die vom Computer gedachte Zahl zu erraten. Für jeden Rateversuch wird dem Benutzer mitgeteilt, ob seine Zahl zu groß oder zu klein ist.

Möglicher Ablauf dieses Programms *zahlrat.c*:

```
Ich denke mir eine Zahl aus dem Intervall [1,x].
Du musst dann versuchen, diese Zahl zu erraten.

Zunächst musst du einmal festlegen, wie gross die zu
ratende Zahl maximal sein darf: 40 (←)

Hm..... OK, ich habe eine Zahl

Dein 1.Versuch: 20 (←) .....zu niedrig
Dein 2.Versuch: 30 (←) .....zu hoch
Dein 3.Versuch: 25 (←) .....zu niedrig
Dein 4.Versuch: 28 (←) .....zu hoch
Dein 5.Versuch: 26 (←) .....zu niedrig
Dein 6.Versuch: 27 (←) .....Richtig

Du hast 6 Versuche zum Erraten der Zahl benötigt.
```

17.2 Quadratwurzel mit NEWTON-Iteration Mathematik(4;<50)

Für eine beliebige Zahl (größer als 0) soll die Quadratwurzel mit einer einzugebenden Genauigkeit angenähert werden. Das klassische NEWTON-Verfahren wird auf die Funktion

$$f(x) = x^2 + z$$

zur näherungsweisen Bestimmung einer Nullstelle angewendet. Mit dem Startwert

$$x_0 = z$$

wird eine neue Näherung

$$x_1 = (x_0 + \frac{z}{x_0})/2$$

gebildet. Das Verfahren wird mit $x_0 = x_1$ solange fortgesetzt, wie folgendes gilt:

$$|x_1 - \frac{z}{x_1}| > k$$

Erstellen Sie ein C-Programm *quadwurz.c*, das zunächst den Wert (x_0) einliest, von dem die Quadratwurzel zu berechnen ist. Danach soll dieses Programm den Wert von k einlesen. Zudem soll der Benutzer noch festlegen können, ob die Zwischenwerte bei dieser Iteration auszugeben sind oder nicht.

Mögliche Abläufe dieses Programms *quadwurz.c*:

```
Zu welcher Zahl soll Quadratwurzel berechnet werden: 2
Wie gro"s darf die Abweichung vom wirklichen Wert maximal sein: 0.00001
Sollen die einzelnen Iterationsschritte angezeigt werden (j/n): j

1.5000000000000000 (1.Iteration)
1.4166666666666667 (2.Iteration)
1.414215686274510 (3.Iteration)

NEWTON-Wert: 1.414215686274510
Wirklicher Wert: 1.414213562373095
-----
Abweichung: 0.000002123901415
```

```
Zu welcher Zahl soll Quadratwurzel berechnet werden: 9
Wie gro"s darf die Abweichung vom wirklichen Wert maximal sein: 0.0000001
Sollen die einzelnen Iterationsschritte angezeigt werden (j/n): n

NEWTON-Wert: 3.000000001396984
Wirklicher Wert: 3.000000000000000
-----
Abweichung: 0.000000001396984
```

```
Zu welcher Zahl soll Quadratwurzel berechnet werden: 9 
Wie gro"s darf die Abweichung vom wirklichen Wert maximal sein: 0.0000000005 
Sollen die einzelnen Iterationsschritte angezeigt werden (j/n): j 

      5.0000000000000000 (1.Iteration)
      3.4000000000000000 (2.Iteration)
      3.023529411764706 (3.Iteration)
      3.000091554131380 (4.Iteration)
      3.000000001396984 (5.Iteration)
      3.0000000000000000 (6.Iteration)

NEWTON-Wert: 3.0000000000000000
Wirklicher Wert: 3.0000000000000000
-----
Abweichung: 0.0000000000000000
```


Kapitel 18

Die break-Anweisung

18.1 Würfelspiel Bis 100 *Allgemein(4;<110)*

Bei diesem Würfelspiel würfelt jeder Spieler sooft er will. Verzichtet ein Spieler auf weiteres Würfeln, bevor er eine 1 würfelt, bekommt er die Summe der bisher erzielten Augen gutgeschrieben. Würfelt er aber eine 1, so erhält er gar nichts und der andere Spieler kommt an die Reihe. Es gewinnt, wer als erster 100 Augen hat.

Erstellen Sie ein C-Programm *bis100.c*, das die Rolle des ersten Spielers übernimmt. Ihr Programm soll dabei nach folgender Strategie spielen: Es würfelt solange, bis seine Punktzahl 19 übersteigt oder bis es 5 mal gewürfelt hat, je nachdem was zuerst eintritt. Da es sich um ein "ehrliches Programm" handelt, übernimmt es auch die Schiedsrichterrolle, indem es immer die momentanen Zwischenstände ausgibt und den Benutzer fragt, ob er noch einmal würfeln möchte. Am Ende eines Spiels soll der momentane Spielstand ausgegeben, und der Benutzer gefragt werden, ob er noch einmal spielen möchte. Wenn ja, so beginnt ein neues Spiel, ansonsten wird der Endstand ausgegeben.

18.2 Primzahlen *Mathematik(4;<40)*

Erstellen Sie ein C-Programm *primzahl.c*, das alle Primzahlen zwischen *m* und *n* ermittelt und ausgibt. *m* und *n* sind dabei einzugeben.

Mögliche Abläufe dieses Programms *primzahl.c* sind z.B.:

```
Primzahlen
=====

Dieses Programm gibt Ihnen alle Primzahlen zwischen
m und n aus. m und n sind dabei einzugeben.
m: 1
n: 500

    2,      3,      5,      7,      11,      13,      17,      19,
   23,     29,     31,     37,     41,     43,     47,     53,
   59,     61,     67,     71,     73,     79,     83,     89,
   97,    101,    103,    107,    109,    113,    127,    131,
  137,    139,    149,    151,    157,    163,    167,    173,
  179,    181,    191,    193,    197,    199,    211,    223,
  227,    229,    233,    239,    241,    251,    257,    263,
  269,    271,    277,    281,    283,    293,    307,    311,
  313,    317,    331,    337,    347,    349,    353,    359,
  367,    373,    379,    383,    389,    397,    401,    409,
  419,    421,    431,    433,    439,    443,    449,    457,
  461,    463,    467,    479,    487,    491,    499,
```

```
.....
m: 12345000
n: 12345500

12345001, 12345017, 12345049, 12345071, 12345083, 12345121, 12345127, 12345143,
12345149, 12345163, 12345169, 12345191, 12345209, 12345211, 12345233, 12345253,
12345259, 12345283, 12345293, 12345301, 12345313, 12345317, 12345323, 12345341,
12345367, 12345371, 12345373, 12345379, 12345397, 12345413, 12345419, 12345427,
12345433, 12345439, 12345479, 12345491, 12345493, 12345499,
```

Kapitel 22

Funktionen

22.1 Allgemeines zu Funktionen

Zu diesem Kapitel sind keine Übungen vorhanden.

22.2 Erstellen eigener Funktionen

22.2.1 Länge eines Streckenzuges *Mathematik(3;<50)*

Erstellen Sie ein Programm *strekzug.c*, das die Länge eines Streckenzuges durch die nacheinander angegebenen Punkte eines Koordinatensystems ermittelt.

Mögliche Abläufe dieses Programms *strekzug.c*:

```
Strecken-Berechnungen
=====

Bitte Startpunkt eingeben (x,y): 5,3 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): 2,4 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): 7,9 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): -1 (↵)

=> Die Streckenlaenge betraegt: 10.23 Einheiten
```

```
Strecken-Berechnungen
=====

Bitte Startpunkt eingeben (x,y): 0,0 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): 4,2 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): 3,-1 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): 1,3 (↵)
Neuer Streckenpunkt x,y (Abbruch mit x=-1): -1 (↵)

=> Die Streckenlaenge betraegt: 12.11 Einheiten
```

22.2.2 Funktionen für binäre Operationen *DV-Wissen(4;<150)*

Zuweilen, besonders bei hardwarenaher Programmierung, ist es nützlich, einzelne Bits in einem Byte überprüfen, setzen, rücksetzen und löschen zu können. Schreiben Sie für jede dieser Operation eine kleine Funktion. Um die Funktionen testen zu können, schreiben Sie ein Programm *bin_op.c*, das zunächst ein Byte im Binärmuster einliest und die zu ändernde Bitposition erfragt. Je nach gewünschter Operation wird dann das Bitmuster entsprechend verändert.

Anmerkung: Aus Zeitgründen werden solche Funktionalitäten auch oft als Makros realisiert. Hier soll aber das Schreiben von Funktionen geübt werden! Ein möglicher Programmablauf von *bin_op.c* wäre:

```
Bitte geben sie 1 Byte als binaere Folge ein: 10000000 ⌨
Auf welche Bitposition soll sich Operation beziehen? (7..0): 2 ⌨
1: Bit pruefen
2: Bit setzen
3: Bit ruecksetzen/loeschen
4: Bit negieren
5: Ende
Ihre Wahl: 2 ⌨
Nach der Bitoperation ergibt sich folgendes Bitmuster fuer das Byte: 10000100
Auf welche Bitposition soll sich Operation beziehen? (7..0): 5 ⌨
1: Bit pruefen
....
Ihre Wahl: 1 ⌨
Das Bit 5 ist im Byte nicht gesetzt
Auf welche Bitposition soll sich Operation beziehen? (7..0): 7 ⌨
....
3: Bit ruecksetzen/loeschen
....
Ihre Wahl: 3 ⌨
Nach der Bitoperation ergibt sich folgendes Bitmuster fuer das Byte: 00000100
Auf welche Bitposition soll sich Operation beziehen? (7..0): 1 ⌨
....
4: Bit negieren
5: Ende
Ihre Wahl: 4 ⌨
Nach der Bitoperation ergibt sich folgendes Bitmuster fuer das Byte: 00000110
Auf welche Bitposition soll sich Operation beziehen? (7..0): 5 ⌨
....
5: Ende
Ihre Wahl: 5 ⌨
```

22.3 Die Parameter von Funktionen

22.3.1 Zeiten-Taschenrechner *Allgemein(4;<120)*

Erstellen Sie ein Programm *zeitrech.c*, das wahlweise zwei Uhrzeiten addiert oder voneinander subtrahiert.

Mögliche Abläufe dieses Programms *zeitrech.c*:

```
Taschenrechner fuer Uhrzeiten
=====
```

```
Bitte Startzeit eingeben (hh:mm:ss): 12:45 ⌨
Bitte 2. Zeit eingeben (hh:mm:ss): 11:40:25 ⌨
Bitte Operation eingeben (+/-): - ⌨

12:45:00 - 11:40:25 = 01:04:35
```

```
Taschenrechner fuer Uhrzeiten
=====
```

```
Bitte Startzeit eingeben (hh:mm:ss): 23:15:12 ⌨
Bitte 2. Zeit eingeben (hh:mm:ss): 5 ⌨
Bitte Operation eingeben (+/-): + ⌨

23:15:12 + 05:00:00 = 1 Tag 04:15:12
```

```
Taschenrechner fuer Uhrzeiten
=====
```

```
Bitte Startzeit eingeben (hh:mm:ss): 00:31:42 ⌨
Bitte 2. Zeit eingeben (hh:mm:ss): 00:45 ⌨
Bitte Operation eingeben (+/-): - ⌨

00:31:42 - 00:45:00 = -23:46:42
```

22.3.2 Sortieren von 4 Zahlen *DV-Wissen(4;<70)*

Erstellen Sie ein Programm *sort4zah.c*, das vier ganze Zahlen einliest und diese aufsteigend sortiert. Der Anwender soll dabei den Vorgang des Sortierens mitverfolgen können, wie z.B.:

1. Ablaufbeispiel:

```
Sortieren von 4 Integer Zahlen
=====
```

```
Zahl1?: 1 ⌨
Zahl2?: 2 ⌨
Zahl3?: 3 ⌨
Zahl4?: 4 ⌨
Was soll das?!
Die Zahlen sind bereits sortiert!!
```

2. Ablaufbeispiel:

```

Sortieren von 4 Integer Zahlen
=====
Zahl1?: 7 (↩)
Zahl2?: 5 (↩)
Zahl3?: 3 (↩)
Zahl4?: 1 (↩)

  1. Durchlauf - Aktueller Stand:
Zahl1: 5
Zahl2: 3
Zahl3: 1
Zahl4: 7
Weiter mit Return... (↩)

  2. Durchlauf - Aktueller Stand:
Zahl1: 3
Zahl2: 1
Zahl3: 5
Zahl4: 7
Weiter mit Return... (↩)

  3. Durchlauf - Aktueller Stand:
Zahl1: 1
Zahl2: 3
Zahl3: 5
Zahl4: 7
Weiter mit Return... (↩)
!!!! FERTIG nach 3 Durchläufen !!!!!

```

3. Ablaufbeispiel:

```

Sortieren von 4 Integer Zahlen
=====
Zahl1?: 1 (↩)
Zahl2?: 4 (↩)
Zahl3?: 7 (↩)
Zahl4?: 2 (↩)

  1. Durchlauf - Aktueller Stand:
Zahl1: 1
Zahl2: 4
Zahl3: 2
Zahl4: 7
Weiter mit Return... (↩)

  2. Durchlauf - Aktueller Stand:
Zahl1: 1
Zahl2: 2
Zahl3: 4
Zahl4: 7
Weiter mit Return... (↩)
!!!! FERTIG nach 2 Durchläufen !!!!!

```

Kapitel 24

Präprozessor-Direktiven

24.1 Konstanten in `limits.h` und `float.h` C–Syntax(3;<70)

In der Headerdatei `limits.h` sind die Mindestwerte für die verschiedenen Ganzzahltypen und in der Headerdatei `float.h` sind die maximalen und minimalen Werte für die unterschiedlichen Gleitpunkttypen definiert, die für Ihren Compiler gelten. Erstellen Sie zwei Programme `limits.c` und `float.c`, die die in `limits.h` bzw. `float.h` definierten Konstanten am Bildschirm ausgeben.

Mögliche Ausgabe durch `limits.c`:

```
CHAR_BIT = 8
SCHAR_MIN = -128
SCHAR_MAX = 127
UCHAR_MAX = 255
CHAR_MIN = -128
CHAR_MAX = 127
MB_LEN_MAX = 6
SHRT_MIN = -32768
SHRT_MAX = 32767
USHRT_MAX = 65535
INT_MIN = -2147483648
INT_MAX = 2147483647
UINT_MAX = 4294967295
LONG_MIN = -2147483648
LONG_MAX = 2147483647
ULONG_MAX = 4294967295
```

Mögliche Ausgabe durch `float.c`:

```
FLT_RADIX = 2
FLT_DIG = 6
DBL_DIG = 15
LDBL_DIG = 18
FLT_MANT_DIG = 24
DBL_MANT_DIG = 53
LDBL_MANT_DIG = 64
FLT_MIN_EXP = -125
DBL_MIN_EXP = -1021
LDBL_MIN_EXP = -16381
FLT_MIN_10_EXP = -37
DBL_MIN_10_EXP = -307
LDBL_MIN_10_EXP = -4931
FLT_MAX_EXP = 128
DBL_MAX_EXP = 1024
LDBL_MAX_EXP = 16384
FLT_MAX_10_EXP = 38
DBL_MAX_10_EXP = 308
LDBL_MAX_10_EXP = 4932
FLT_MIN = 1.1754944e-38
DBL_MIN = 2.225073858507201e-308
LDBL_MIN = 3.3621031431120935063e-4932
FLT_MAX = 3.4028235e+38
DBL_MAX = 1.797693134862316e+308
LDBL_MAX = 1.189731495357231765e+4932
FLT_EPSILON = 1.1920929e-07
DBL_EPSILON = 2.220446049250313e-16
LDBL_EPSILON = 1.084202172485504434e-19
FLT_ROUNDS = 1
```


Kapitel 25

Zeiger und Arrays

25.1 Eindimensionale Arrays

25.1.1 Ziehen der Lottozahlen simulieren *Allgemein(4;<40)*

Erstellen Sie ein C-Programm *lotto.c*, das zufällig x verschiedene Zahlen aus einem Bereich von 1 bis n ermittelt. x und n sollen dabei eingegeben werden.

Mögliche Abläufe dieses Programms *lotto.c*:

```
Lottozahlen-Simulation
=====
Wieviele Kugeln sollen zur Verfuegung stehen (mind. 1 und max 100): 49
Wieviele werden davon gezogen (mind. 1 und max. 49): 6

==== 6 aus 49 ====
12    15    24    32    37    40
```

```
Lottozahlen-Simulation
=====
Wieviele Kugeln sollen zur Verfuegung stehen (mind. 1 und max 100): 7
Wieviele werden davon gezogen (mind. 1 und max. 7): 7

==== 7 aus 7 ====
1     2     3     4     5     6     7
```

```
Lottozahlen-Simulation
=====
Wieviele Kugeln sollen zur Verfuegung stehen (mind. 1 und max 100): 50
Wieviele werden davon gezogen (mind. 1 und max. 50): 12

==== 12 aus 50 ====
10    13    23    25    27    29    33    36    37    41    42    43
```

25.1.2 Primzahlen mit dem Sieb des Eratosthenes *Mathematik(3;<20)*

Es sollen alle Primzahlen zwischen 1 und 1000 bestimmt werden. Dazu soll das sogenannte *Sieb des Eratosthenes* verwendet werden, das folgende Vorgehensweise vorschreibt: Zunächst wird auf die Zahl 2 positioniert, und dann alle Vielfachen von 2 herausgestrichen, was nachfolgend durch Unterstreichen der betreffenden Zahlen gezeigt ist:

```

1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 ...
   ^  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   |

```

Im nächsten Schritt wird um eine Zahl weiterpositioniert, in unserem Fall also auf 3; ist diese Zahl noch vorhanden, so handelt es sich um eine Primzahl, von der wiederum alle Vielfachen zu streichen sind:

```

1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 ...
   ^  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
   |

```

Die nächste Primzahl nach 3 wäre dann 5 (noch nicht gestrichen), von der wieder alle Vielfachen zu streichen sind usw. Dieses Verfahren wird wiederholt, bis 1000 erreicht ist.

Ablauf des Programms *primsieb.c*:

Die Primzahlen von 1 bis 1000 sind:									
2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	71
73	79	83	89	97	101	103	107	109	113
127	131	137	139	149	151	157	163	167	173
179	181	191	193	197	199	211	223	227	229
233	239	241	251	257	263	269	271	277	281
283	293	307	311	313	317	331	337	347	349
353	359	367	373	379	383	389	397	401	409
419	421	431	433	439	443	449	457	461	463
467	479	487	491	499	503	509	521	523	541
547	557	563	569	571	577	587	593	599	601
607	613	617	619	631	641	643	647	653	659
661	673	677	683	691	701	709	719	727	733
739	743	751	757	761	769	773	787	797	809
811	821	823	827	829	839	853	857	859	863
877	881	883	887	907	911	919	929	937	941
947	953	967	971	977	983	991	997		

25.1.3 Das Ziegenproblem Wahrscheinlichkeitstheorie(4;<40)

Die amerikanische Journalistin *Marilyn vos Savant*, die als die Frau mit dem höchsten Intelligenzquotienten gilt, stellte in einer Zeitschrift folgende Denksportaufgabe:

Sie nehmen an einer Spielshow im Fernsehen teil, bei der Sie eine von drei verschlossenen Türen auswählen sollen. Hinter einer Tür wartet der Preis, ein Auto, hinter den beiden anderen stehen Ziegen. Sie zeigen auf eine Tür, sagen wir die Nummer eins. Sie bleibt vorerst geschlossen. Der Moderator weiß, hinter welcher Tür sich das Auto befindet; mit den Worten: "Ich zeige Ihnen was" öffnet er eine andere Tür, zum Beispiel Nummer drei, und eine meckernde Ziege schaut ins Publikum. Er fragt: "Bleiben Sie bei Nummer eins, oder wählen Sie Nummer zwei?"

Sollten Sie die Tür wechseln oder bei ihrer ursprünglichen Wahl bleiben? Die meisten Leute antworten auf diese Frage: "Es steht fifty-fifty, also ist es egal, ob man wechselt oder nicht". Ist diese Antwort falsch oder richtig?

Erstellen Sie ein Programm *ziegprob.c*, das diese Spielshow simuliert und die unterschiedlichen Gewinnchancen bei einem Wechsel bzw. bei einem Nichtwechsel berechnet und ausgibt, wie z.B.

```
Das Ziegenproblem
=====

Wieviele Simulationen: 10000 (↔)

-----

Moderator darf weder Rate- noch Auto-Tür öffnen
Gewinnchance beim Wechseln:      .....
                               beim Nicht-Wechseln: .....

-----
```

25.2 Mehrdimensionale Arrays

25.2.1 Matrizenmultiplikation *Mathematik(4;<100)*

Erstellen Sie ein Programm *matmult.c*, das zwei Matrizen multipliziert. Um zwei Matrizen miteinander multiplizieren zu können, muss die Spaltenzahl der ersten Matrix gleich der Zeilenzahl der zweiten Matrix sein. Wenn wir zwei Matrizen haben:

$$a = \begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1m} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2m} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3m} \\ \dots & \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nm} \end{pmatrix}$$

$$b = \begin{pmatrix} b_{11} & b_{12} & b_{13} & \dots & b_{1k} \\ b_{21} & b_{22} & b_{23} & \dots & b_{2k} \\ b_{31} & b_{32} & b_{33} & \dots & b_{3k} \\ \dots & \dots & \dots & \dots & \dots \\ b_{m1} & b_{m2} & b_{m3} & \dots & b_{mk} \end{pmatrix}$$

dann wird die Multiplikation für $c = a \cdot b$ nach folgendem Algorithmus gebildet:

1. Die Matrix c hat zunächst einmal n Zeilen und k Spalten.
2. Jedes Element der Matrix c wird wie folgt berechnet:

$$\begin{aligned} c_{11} &= a_{11} \cdot b_{11} + a_{12} \cdot b_{21} + a_{13} \cdot b_{31} + \dots + a_{1m} \cdot b_{m1} \\ c_{21} &= a_{21} \cdot b_{11} + a_{22} \cdot b_{21} + a_{23} \cdot b_{31} + \dots + a_{2m} \cdot b_{m1} \\ &\dots \dots \dots \\ c_{12} &= a_{11} \cdot b_{12} + a_{12} \cdot b_{22} + a_{13} \cdot b_{32} + \dots + a_{1m} \cdot b_{m2} \\ &\dots \dots \dots \\ c_{nk} &= a_{n1} \cdot b_{1k} + a_{n2} \cdot b_{2k} + a_{n3} \cdot b_{3k} + \dots + a_{nm} \cdot b_{mk} \end{aligned}$$

Abbildung 25.1 soll dieses Verfahren nochmals veranschaulichen.

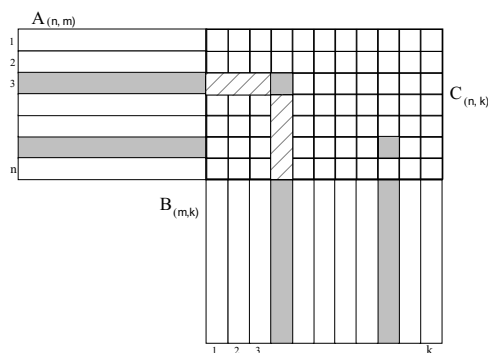


Abbildung 25.1: Multiplikation von zwei Matrizen

Möglicher Ablauf des Programms *matmult.c*:

```

Zeilen,Spalten der 1. Matrix: 2,3 
Zeilen,Spalten der 2. Matrix: 3,4 

Eingabe der 1. Matrix
Element 1,1: 1 
Element 1,2: 2 
Element 1,3: 3 
Element 2,1: 4 
Element 2,2: 5 
Element 2,3: 6 

Eingabe der 2. Matrix
Element 1,1: 7 
Element 1,2: 8 
Element 1,3: 2.3 
Element 1,4: 1 
Element 2,1: 2 
Element 2,2: 3 
Element 2,3: 2 
Element 2,4: 2 
Element 3,1: 4 
Element 3,2: 3 
Element 3,3: 5 
Element 3,4: 4 

```

1. Matrix

1.00	2.00	3.00
4.00	5.00	6.00

2. Matrix

7.00	8.00	2.30	1.00
2.00	3.00	2.00	2.00
4.00	3.00	5.00	4.00

Ergebnismatrix

23.00	23.00	21.30	17.00
62.00	65.00	49.20	38.00

25.2.2 Das Spiel Tictac *Allgemein(4;<150)*

Auf einem quadratischen Spielfeld mit $n \times n$ Feldern setzen zwei Spieler abwechselnd je einen Stein ihrer Farbe (X oder O). Wer zuerst 3 Steine in einer Reihe, Spalte oder Diagonale setzen kann, gewinnt das Spiel. Erstellen Sie ein C-Programm *tictac.c*, das dieses Spiels simuliert und schließlich den Gewinner ausgibt, wie z.B.:

```

Spielfeldgroesse (mind. 3, maximal 10) ? 5
Dein Zug, Spieler 1 (Zeile,Spalte) ? 2,3
1 . . . . .
2 . . X . .
3 . . . . .
4 . . . . .
5 . . . . .
Dein Zug, Spieler 2 (Zeile,Spalte) ? 3,4
1 . . . . .
2 . . X . .
3 . . . O .
4 . . . . .
5 . . . . .
Dein Zug, Spieler 1 (Zeile,Spalte) ? 2,4
1 . . . . .
2 . . X X .
3 . . . O .
4 . . . . .
5 . . . . .
Dein Zug, Spieler 2 (Zeile,Spalte) ? 2,3
....Unerlaubter Spielzug
Dein Zug, Spieler 2 (Zeile,Spalte) ? 2,2
1 . . . . .
2 . O X X .
3 . . . O .
4 . . . . .
5 . . . . .
Dein Zug, Spieler 1 (Zeile,Spalte) ? 2,5
1 . . . . .
2 . O X X X
3 . . . O .
4 . . . . .
5 . . . . .
Spieler 1 hat gewonnen

```

25.2.3 Game of Life (Beispiel für zellulare Automaten) Allgemein(5;<150)

Die wesentliche Eigenschaft lebender Organismen ist ihre Fähigkeit zur Selbstreproduktion. Jeder Organismus kann Nachkommen erzeugen, die – bis auf Feinheiten – eine Kopie des erzeugenden Organismus sind. *John von Neumann* stellte folgende Frage: *Sind auch Maschinen (z.B. Roboter) zur Selbstreproduktion fähig? Welche Art logischer Organisation ist dafür notwendig und hinreichend?* S. M. Ulam schlug die Verwendung sogenannter *zellulärer Automaten* vor. Einen zellularen Automaten kann man sich anschaulich als eine in Quadrate (Zellen) aufgeteilte Ebene vorstellen. Auf jedem Quadrat befindet sich ein endlicher Automat, dessen Verhalten von seinem eigenen Zustand und von den Zuständen gewisser Nachbarn (Zellen) abhängt. Alle Automaten sind gleich und arbeiten im gleichen Takt.

Ein berühmtes Beispiel für einen zellularen Automaten ist das *Game of Life (Lebensspiel)* des englischen Mathematikers *John H. Conway*. Jede Zelle hat zwei Zustände (lebend, tot) und die Umgebung der Zelle besteht aus den angrenzenden acht Nachbarquadraten. Die Zeit verstreicht in diskreten Schritten. Von einem Schlag der kosmischen Uhr bis zum nächsten verharrt die Zelle im zuvor eingenommenen Zustand, beim Gong aber wird nach den folgenden Regeln erneut über Leben und Tod entschieden:

❑ **Geburt**

Eine tote Zelle feiert Auferstehung, wenn drei ihrer acht Nachbarn leben.

❑ **Tod durch Überbevölkerung**

Eine Zelle stirbt, wenn vier oder mehr Nachbarn leben.

❑ **Tod durch Vereinsamung**

Eine Zelle stirbt, wenn sie keinen oder nur einen lebenden Nachbarn hat.

Eine lebende Zelle bleibt also genau dann am Leben, wenn sie zwei oder drei lebende Nachbarn besitzt. Der Reiz dieses Spiels liegt in seiner Unvorhersehbarkeit. Nach den oben angegebenen Regeln kann eine Population aus lebenden Zellen grenzenlos wachsen, sich zu einem periodisch wiederkehrenden oder stabilen Muster entwickeln oder aussterben.

Erstellen Sie ein Programm *life.c*, bei dem der Benutzer zunächst die Länge und Breite des Spielfelds eingibt. Danach soll der Benutzer wählen können, ob er die Anfangspopulation selbst (über Mausklicks) eingeben will oder ob diese zufällig sein soll. Im zweiten Fall soll der Benutzer noch eingeben können, wie viele Zellen zu Beginn leben sollen.

Die Abbildungen 25.2, 25.3 und 25.4 zeigen mögliche Anzeigen des Programms *life.c*.

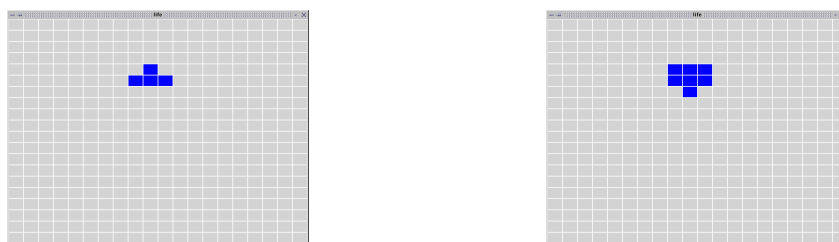


Abbildung 25.2: Population zu Beginn (links) und nach dem ersten Schritt (rechts)

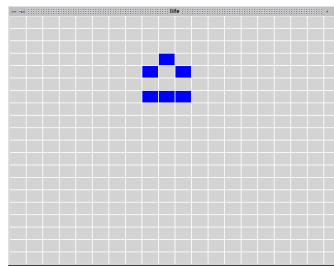


Abbildung 25.3: Population nach zwei Schritten

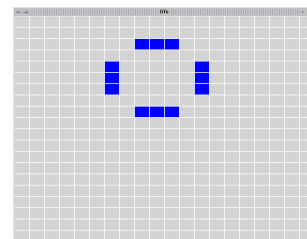
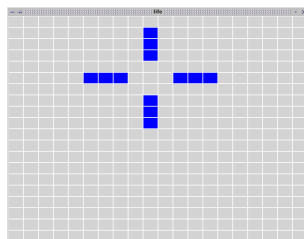


Abbildung 25.4: Population nach einigen weiteren Schritten, die sich nun ständig wiederholen

Weiter interessante Figuren sind:

			x	x
			xxxx	
xx	x	x	x	x
xx	x	x	x xx x	
x	x	xxx	x x	
			xxxx	
Pentomino	Kreuz	Gleiter	Cheshire-Katze	^M

Eine andere Population mit überraschendem Verhalten ist:

```
xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx xxxxx
```

Index

aepfel.c, 35	kandproz.c, 24	quadwurz.c, 42
benzinv.c, 19	kettenpi.c, 12	quersum.c, 22, 39
bin_op.c, 48	kreis.c, 30	
bis100.c, 45	kugel.c, 17	rahmen.c, 22
bytedual.c, 15		
dualbcd.c, 39	leibniz.c, 32	schalt.c, 25
erdumf.c, 18	life.c, 59	sort4zah.c, 49
exporeih.c, 32	limits.c, 51	strekzug.c, 47
	ljahrkm.c, 17	sumunger.c, 31
	lotto.c, 53	
		teilbar.c, 26
faktor.h, 21	matmult.c, 56	teiler.c, 33
fallzeit.c, 18	menue.c, 3, 27	tictac.c, 58
falschir.c, 23	minmax.c, 37	
fibo.c, 37		vchiffre.c, 36
flaeche.c, 28	naechtag.c, 29	vumrech.c, 21
float.c, 51	okdezhex.c, 22	
gruss.c, 26		zahlrat.c, 41
harmon.c, 31	party.c, 33	zeitrech.c, 49
	primsieb.c, 54	ziegprob.c, 55
	primzahl.c, 46	zinskapi.c, 19