

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

# **Übungen zur Programmiersprache C**

**für das Praktikum  
zu Informatik 2**

# Inhaltsverzeichnis

<b>0</b>	<b>Einleitung</b>	<b>1</b>
0.1	Informationen zu den einzelnen Übungen . . . . .	1
<b>21</b>	<b>Graphikprogrammierung unter Linux</b>	<b>3</b>
21.1	Der Weg einer Billard-Kugel <i>Allgemein(4;&lt;100)</i> . . . . .	3
21.2	PI mit Regentropfen berechnen <i>Mathematik(4;&lt;100)</i> . . . . .	4
21.3	Spirolateralkurven <i>Mathematik(5;&lt;120)</i> . . . . .	5
21.4	Simulation einer analogen Uhr <i>Allgemein(6;&lt;160)</i> . . . . .	6
21.5	Eine dreidimensionale Uhr <i>Allgemein(6;&lt;160)</i> . . . . .	6
21.6	Dreidimensionale Funktion $z = x^2 - y^2$ <i>Mathematik(5;&lt;100)</i> . . . . .	7
21.7	Katze jagt die Maus <i>Physik(5;&lt;110)</i> . . . . .	9
21.8	Basketball spielen <i>Physik(6;&lt;150)</i> . . . . .	11
21.9	Spiralen über Kreise <i>Mathematik(5;&lt;100)</i> . . . . .	12
21.10	Kreis mit putpixel() zeichnen <i>Mathematik(4;&lt;30)</i> . . . . .	12
21.11	Kreise mit Linien zeichnen <i>Mathematik(5;&lt;100)</i> . . . . .	13
21.12	Der Ellipsenwurm <i>Mathematik(5;&lt;70)</i> . . . . .	14
21.13	Huhn im dreieckigen Kornfeld <i>Fraktale(4;&lt;120)</i> . . . . .	15
21.14	Die Kochsche Schneeflocke <i>Fraktale(6;&lt;100)</i> . . . . .	16
21.15	Die Mandelbrot-Menge <i>Chaos(6;&lt;250)</i> . . . . .	17
21.16	Ein Malprogramm <i>Zeichnen(6;&lt;250)</i> . . . . .	21
<b>22</b>	<b>Funktionen</b>	<b>23</b>
22.1	Allgemeines zu Funktionen . . . . .	23
22.2	Erstellen eigener Funktionen . . . . .	23
22.2.1	Länge eines Streckenzuges <i>Mathematik(3;&lt;50)</i> . . . . .	23
22.2.2	Funktionen für binäre Operationen <i>DV-Wissen(4;&lt;150)</i> . . . . .	24

22.3	Die Parameter von Funktionen	25
22.3.1	Zeiten-Taschenrechner <i>Allgemein(4;&lt;120)</i>	25
22.3.2	Sortieren von 4 Zahlen <i>DV-Wissen(4;&lt;70)</i>	25
22.4	Ellipsen-Prototypen	27
22.4.1	Maximum aus vielen Zahlen <i>Allgemein(3;&lt;40)</i>	27
22.5	Rekursive Funktionen	27
22.5.1	Binomialkoeffizient <i>Mathematik(3;&lt;50)</i>	27
22.5.2	Größter gemeinsamer Teiler <i>Mathematik(4;&lt;40)</i>	28
22.5.3	Umwandeln einer Dezimalzahl in Dualzahl <i>DV-Wissen(3;&lt;30)</i>	28
22.5.4	Quersumme und Umdrehen einer Zahl <i>Mathematik(3;&lt;40)</i>	29
22.5.5	Das Terrassenproblem <i>Allgemein(4;&lt;30)</i>	30
22.6	Zeiger auf Funktionen	31
22.6.1	Der Baum des Pythagoras <i>Allgemein(6;&lt;100)</i>	31
<b>23</b>	<b>Speicherklassen und Modultechnik</b>	<b>35</b>
23.1	Gültigkeitsbereich, Lebensdauer, Speicherort	35
23.1.1	Ausgabe des Programms block.c <i>C-Syntax(3;&lt;10)</i>	35
23.2	Schlüsselwörter extern, auto, static und register	36
23.2.1	Ausgabe des Programms speikla1.c <i>C-Syntax(3;&lt;10)</i>	36
23.2.2	Ausgabe des Programms speikla2.c <i>C-Syntax(4;&lt;10)</i>	37
23.2.3	Ausgabe des Programms speikla3.c <i>C-Syntax(5;&lt;10)</i>	38
23.3	Schlüsselwörter const und volatile	39
23.3.1	Konstante Zeiger und Zeiger auf Konstanten <i>C-Syntax(4;&lt;10)</i>	39
23.3.2	const-Parameter bei Funktionsdefinitionen <i>C-Syntax(4;&lt;10)</i>	39
23.4	Die Modultechnik	40
23.4.1	Turingmaschine zur binären Addition von 1 <i>Informatik(5;&lt;200)</i>	40
23.4.2	Verdopplungs-Turingmaschine <i>Informatik(5;&lt;220)</i>	40
23.4.3	Multiplizier-Turingmaschine <i>Informatik(6;&lt;300)</i>	40
<b>24</b>	<b>Präprozessor-Direktiven</b>	<b>41</b>
24.1	Konstanten in limits.h und float.h <i>C-Syntax(3;&lt;70)</i>	41
<b>25</b>	<b>Zeiger und Arrays</b>	<b>43</b>
25.1	Eindimensionale Arrays	43
25.1.1	Ziehen der Lottozahlen simulieren <i>Allgemein(4;&lt;40)</i>	43
25.1.2	Primzahlen mit dem Sieb des Eratosthenes <i>Mathematik(3;&lt;20)</i>	44

25.1.3	Das Ziegenproblem	<i>Wahrscheinlichkeitstheorie(4;&lt;40)</i>	45
25.2	Mehrdimensionale Arrays		46
25.2.1	Matrizenmultiplikation	<i>Mathematik(4;&lt;100)</i>	46
25.2.2	Das Spiel Tictac	<i>Allgemein(4;&lt;150)</i>	48
25.2.3	Game of Life (Beispiel für zellulare Automaten)	<i>Allgemein(5;&lt;150)</i>	49
25.3	Zusammenhänge zwischen Arrays und Zeigern		51
25.3.1	Gaußsche Eliminationsverfahren	<i>Mathematik(5;&lt;150)</i>	51
25.3.2	Folgen von Nullen und Einsen	<i>Mathematik(4;&lt;50)</i>	54
25.3.3	Modus (Modalwert) einer Zahlenreihe	<i>Statistik(4;&lt;80)</i>	54
25.4	Strings und char-Zeiger		55
25.4.1	Streichen eines Zeichens aus String	<i>Allgemein(4;&lt;40)</i>	55
25.4.2	Ersetzen von Wörtern in einem String	<i>Allgemein(5;&lt;80)</i>	55
25.4.3	Palindrome in einem Text finden	<i>Allgemein(4;&lt;50)</i>	56
25.4.4	Sichere Zahleneingabe	<i>Allgemein(5;&lt;150)</i>	56
25.4.5	Wortlängen in einem Text	<i>Allgemein(3;&lt;40)</i>	57
25.5	Array-Initialisierungen		58
25.5.1	Irren ist käferlich	<i>Allgemein(4;&lt;80)</i>	58
25.5.2	Wochentag zu bestimmten Datum	<i>Allgemein(4;&lt;50)</i>	59
25.5.3	Tagesnummer zu Datum und umgekehrt	<i>Allgemein(4;&lt;70)</i>	60
25.6	Zeigerarrays und Zeiger auf Zeiger		61
25.6.1	Interessante Wort-Zahlenfolgen	<i>Allgemein(4;&lt;50)</i>	61
25.6.2	Das d'Hondtsche Höchstzählverfahren	<i>Allgemein(4;&lt;80)</i>	62
25.6.3	Finden von Zahlwörtern in Strings	<i>Allgemein(4;&lt;50)</i>	64
25.6.4	Mischen von Farben	<i>Allgemein(4;&lt;50)</i>	64
<b>26</b>	<b>Argumente auf der Kommandozeile</b>		<b>65</b>
26.1	Dezimalzahlen nach Dual, Oktal und Hexa	<i>Informatik(4;&lt;80)</i>	65
26.2	Rechnen mit Dualzahlen	<i>Informatik(4;&lt;60)</i>	66
<b>27</b>	<b>Dynamische Speicher-Reservierung und -Freigabe</b>		<b>67</b>
27.1	Numerierte oder rückwärtige Textausgabe	<i>Informatik(5;&lt;60)</i>	67
27.2	Strichliste	<i>Informatik(4;&lt;40)</i>	67
27.3	Berechnen von Primzahlen	<i>Mathematik(4;&lt;60)</i>	68
<b>28</b>	<b>Strukturen</b>		<b>69</b>
28.2	Operationen mit Strukturen		69

28.2.1	Suchen eines optimalen Hauses	<i>Allgemein(5;&lt;200)</i>	69
28.2.2	Addieren von deutschen Kommazahlen	<i>Allgemein(5;&lt;80)</i>	71
28.4	Strukturarrays		72
28.4.1	Text in Morse-Code wandeln und umgekehrt	<i>Allgemein(4;&lt;120)</i>	72
28.4.2	Zählen der Schlüsselwörter in C-Programm	<i>Allgemein(3;&lt;50)</i>	72
28.4.3	Umwandeln arabischer in römische Zahlen	<i>Allgemein(4;&lt;50)</i>	73
28.4.4	Notenübersicht und Notenspiegel	<i>Allgemein(3;&lt;60)</i>	74
28.5	Strukturen als Funktionsparameter		76
28.5.1	Tagesdifferenz zwischen zwei Dati	<i>Allgemein(5;&lt;60)</i>	76
28.5.2	Addition von zwei Zeiten	<i>Allgemein(3;&lt;60)</i>	76
28.6	Zeiger und Strukturen		77
28.6.1	Mischen von zwei sortierten Listen	<i>Informatik(4;&lt;150)</i>	77
28.6.2	Eine Liste, aber unterschiedlich sortiert	<i>Informatik(4;&lt;120)</i>	79
28.6.3	Wortstatistik zu einem Text	<i>Allgemein(4;&lt;80)</i>	81
28.6.4	Realisierung eines Binärbaums mit Array	<i>Informatik(4;&lt;100)</i>	82
28.7	Spezielle Strukturen (Unions und Bitfelder)		84
28.7.1	Partnersuche in einem Heiratsinstitut	<i>Allgemein(4;&lt;200)</i>	84
<b>29</b>	<b>Eigene Datentypen</b>		<b>89</b>
29.1	Mischtable aus der Chemie	<i>Chemie(4;&lt;60)</i>	89
<b>30</b>	<b>Dateien</b>		<b>91</b>
30.1	Höhere E/A-Funktionen		91
30.1.1	Ausgeben der letzten n Zeilen einer Datei	<i>Allgemein(4;&lt;60)</i>	91
30.1.2	Ausgeben einer Datei mit Zeilennumerierung	<i>Allgemein(3;&lt;30)</i>	92
30.1.3	Erzeugen von Adress-Aufkleber	<i>Allgemein(3;&lt;50)</i>	92
30.1.4	Suchen von Strings in Dateien	<i>Allgemein(3;&lt;30)</i>	93
30.1.5	Ausgeben bestimmter Zeilen einer Datei	<i>Allgemein(6;&lt;100)</i>	94
30.2	Elementare E/A-Funktionen		94
30.2.1	Anhängen einer Datei an eine andere	<i>Allgemein(3;&lt;40)</i>	94
30.2.2	Rückwärtiges Ausgeben einer Datei	<i>Allgemein(5;&lt;60)</i>	94

# 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:

<b>Ausgabe eines Menüs</b>	<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 21

## Graphikprogrammierung unter Linux

### 21.1 Der Weg einer Billard-Kugel *Allgemein(4;<100)*

Erstellen Sie C-Programm *billard.c*, das den Weg einer Billard-Kugel graphisch am Bildschirm anzeigt. Zunächst muss der Startpunkt der Kugel eingegeben werden. Vom Startpunkt aus bewegt sich die Kugel immer zunächst 45 Grad nach rechts unten. Trifft die Kugel auf den Rand, so wird sie "elastisch" zurückgestoßen; das heißt: der Eintreffwinkel ist gleich dem Abstoßwinkel. Der Weg der Kugel ändert bei diesem Programm immer nach  $n$  Pixel-Ausgaben seine Farbe.  $n$  soll dabei der Benutzer eingeben. Wenn Sie etwas überlegen, werden Sie feststellen, dass sie hier keinerlei mathematischen Funktionen benötigen. Mögliches Aussehen des Billard-Fensters zu verschiedenen Zeitpunkten zeigen die Abbildungen 21.1 und 21.2.

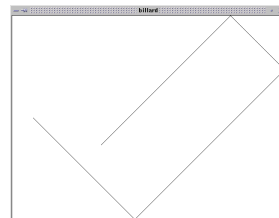
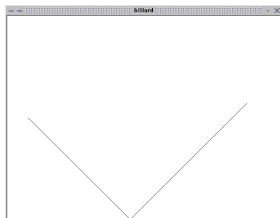


Abbildung 21.1: Billardfenster-Schnappschuß ziemlich am Anfang und kurz danach

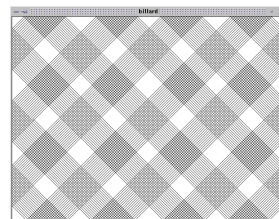
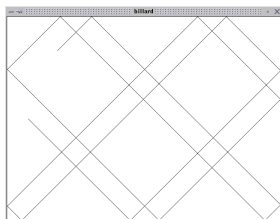


Abbildung 21.2: Billardfenster-Schnappschüsse etwas später



## 21.2 PI mit Regentropfen berechnen *Mathematik(4;<100)*

Erstellen Sie ein C-Programm *piregen.c*, das zunächst ein rotes Quadrat im Graphikfenster malt. In dieses Quadrat soll es dann einen genau passenden Kreis malen. Dieser Kreis soll z.B. blau ausgemalt sein. Nun lassen Sie zufällig Regentropfen in das Quadrat fallen. Das Fallen eines Regentropfes kann dadurch angezeigt werden, dass an der entsprechenden (zufällig ermittelten Position) ein Pixel mit einer bestimmten Farbe gezeichnet wird. Dabei sollte für außerhalb des Kreises gefallene Regentropfen eine andere Farbe verwendet werden, wie für Tropfen im Kreis. Zählt man nun immer die Regentropfen, die in den Kreis fielen und teilt diese Zahl durch die Gesamtzahl der bisher gefallenen Regentropfen, so ergibt sich eine Näherung für  $\frac{\pi}{4}$ . Nimmt man also das Ergebnis mal 4, so hat man eine Näherung für die Zahl  $\pi$ . Es wäre schön, wenn Ihr Programm alle 1000 Tropfen die bisher ermittelte  $\pi$ -Näherung als Zwischeninformation am rechten Bildschirmrand anzeigen würde; siehe auch Abbildung 21.3. Wie viele Regentropfen fallen sollen, muss der Benutzer vor dem Beginn einer Simulation eingeben. Nach der Beendigung einer Simulation wird dem Benutzer die ermittelte Zahl  $\pi$  angezeigt, wobei er auch gefragt wird, ob er eine weitere Simulation wünscht; siehe auch Abbildung 21.4.

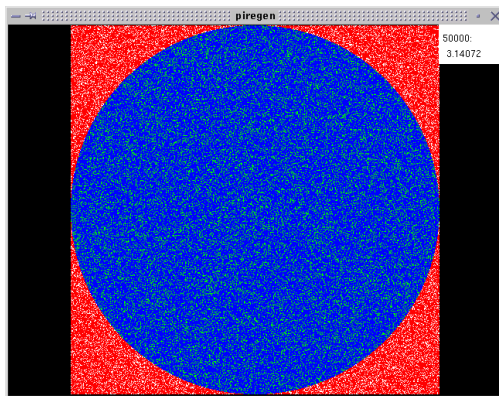


Abbildung 21.3: Ermitteln der Zahl  $\pi$  durch das Fallen von Regentropfen

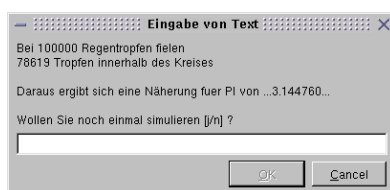


Abbildung 21.4: Anzeigen der ermittelten Zahl  $\pi$  mit Frage, ob weitere Simulation gewünscht

## 21.3 Spirolateralkurven *Mathematik(5;<120)*

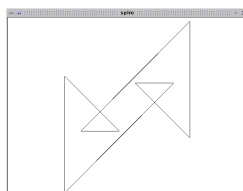
Bei Spirolateralkurven wird eine Figur durch Wiederholungen von bestimmten Motiven gezeichnet. Ein Motiv ist durch die Parameter WINKEL und ANZAHL, die einzugeben sind, festgelegt. Ein Motiv wird dabei nach folgenden Regeln gebildet:

1. Ausgehend vom momentanen Punkt wird um den WINKEL (in mathematisch positive Richtung) gedreht. Dann wird in diese Richtung eine Linie mit Länge 1 gezeichnet. Danach wird dieser Vorgang wiederholt, aber mit doppelter Länge, beim dritten mal mit dreifacher Länge usw., so oft dies die eingegebene ANZAHL verlangt. Damit ist ein Motiv gezeichnet. Insgesamt wurde somit um den Winkel  $ANZAHL * WINKEL$  gedreht.
2. Nun wird das Motiv mit der neuen Anfangsrichtung wiederholt. Die gesamte Figur ist fertig, wenn die ursprüngliche Startrichtung wieder erreicht ist, wenn also der gesamte Drehwinkel ein Vielfaches von 360 Grad ist

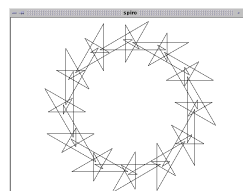
Erstellen Sie ein C-Programm *spiro.c*, das die Werte für ANZAHL und WINKEL einliest, und dann die zugehörige Spirolateralkurve graphisch ausgibt.

Damit die Spirolateralkurve auf den Bildschirm paßt, müssen Sie zuvor einen Probelauf mit den angegebenen Parametern durchführen, um so die maximalen und minimalen x- und y-Werte zu ermitteln, bevor Sie dann in einem neuen Durchgang das Bild entsprechend in das Fenster zeichnen. Die folgenden Parameterbeispiele für (WINKEL, ANZAHL) ergeben besonders schöne oder überraschende Figuren:

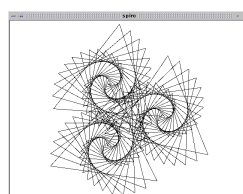
```
(35,3), (66,8), (70,13), (75,90), (86,7), (87,13), (111,3), (124,60), (130,1),
(132,8), (135,7), (150,7), (170,3), (177,15), (188,5), (200,34), (270,89), (300,41)
```



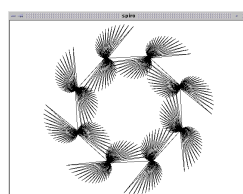
WINKEL=135  
ANZAHL=4



WINKEL=150  
ANZAHL=7



WINKEL=124  
ANZAHL=60



WINKEL=177  
ANZAHL=45

## 21.4 Simulation einer analogen Uhr *Allgemein(6;<160)*

Erstellen Sie ein C-Programm *uhr.c*, das eine analoge Uhr simuliert, wie dies z.B. in Abbildung 21.5 gezeigt ist. Der Benutzer soll entscheiden können, ob er die Start-Uhrzeit selbst eingeben möchte oder ob als Startzeit die momentane Rechner-Uhrzeit vom Programm verwendet werden soll.

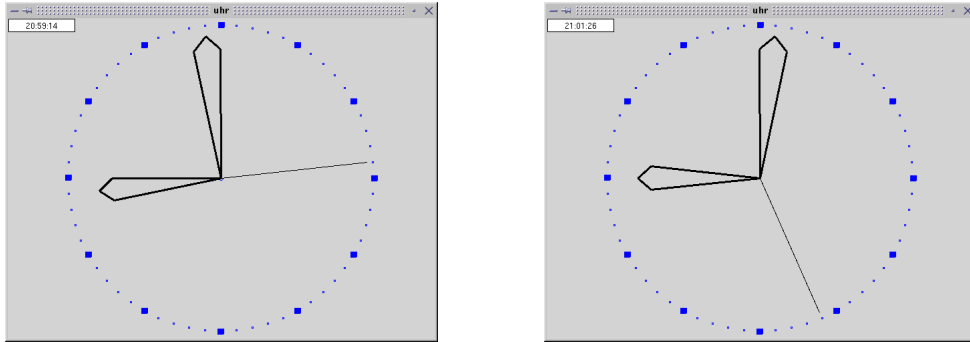


Abbildung 21.5: Simulation einer analogen Uhr

## 21.5 Eine dreidimensionale Uhr *Allgemein(6;<160)*

Erstellen Sie ein C-Programm *uhr3d.c*, das eine dreidimensionale analoge Uhr simuliert, wie dies z.B. in Abbildung 21.6 gezeigt ist.



Abbildung 21.6: Simulation einer dreidimensionalen analogen Uhr

## 21.6 Dreidimensionale Funktion $z = x^2 - y^2$ Mathematik(5;<100)

Erstellen Sie ein C-Programm *dreidim.c*, das die Funktion  $z = x^2 - y^2$  dreidimensional in einem Fenster anzeigt. Falls Sie Schwierigkeiten haben sollten, dann könnten Sie z.B. den nachfolgenden Algorithmus verwenden:

1. Einlesen eines Startwerts, eines Endwerts und einer Schrittweite für x.
2. Einlesen eines Startwerts, eines Endwerts und einer Schrittweite für y.
3. Negieren der beiden Schrittweiten.
4. Dieser 4. Schritt wird in C-Pseudocode angegeben:

```
for (i=1 ; i<=2 ; i++) {
    for (y=yende ; y>=ystart ; y+=yschrittweite) {
        for (x=xende ; x>=xstart ; x+=xschrittweite) {
            z=x2-y2
            xkoord=(x*40+y*16*cos(PI/8)-xmin)*faktorx
            ykoord=(z*10+y*16*sin(PI/8)-ymin)*faktory
            putpixel(xkoord, ykoord, 15)
        }
    }
    if (i==1)
        xschrittweite *= 10
        yschrittweite /= 10
}
```

Die Variablen `xmin`, `ymin`, `faktorx` und `faktory` sorgen dafür, dass das entsprechende Bild genau in das Fenster projiziert wird. Da die Werte dieser Variablen von den jeweils eingegebenen Werten abhängig sind, müssen diese in einem vorherigen Probelauf mit dem gleichen Algorithmus bestimmt werden. Bei diesem 1. Durchgang sollten jedoch noch keine Pixel ausgegeben werden, sondern nur die kleinste und größte x-Koordinate (`xmin` und `xmax`) und die kleinste und größte y-Koordinate (`ymin` und `ymax`) ermittelt werden. Nach diesem Probelauf kann dann `faktorx` und `faktory` wie folgt bestimmt werden:

```
faktorx = getmaxx() / (xmax-xmin)
faktory = getmaxy() / (ymax-ymin)
```

Erst danach beginnt die eigentliche Ausgabe mit dem oben gezeigten Algorithmus.

Abbildung 21.7 zeigt das von Programm *dreidim.c* angezeigte Bilder für die folgenden Werte:

```

Startwert für x: -10
Endwert für x: 10
Schrittweite für x (>0): 0.1
Startwert für y: -20
Endwert für y: 30
Schrittweite für y (>0): 0.3

```

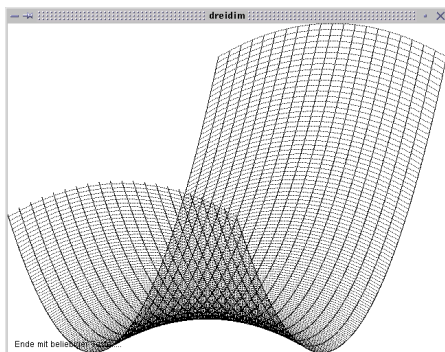


Abbildung 21.7: Dreidimensionaler Plot für die Funktion  $z = x^2 - y^2$

Abbildung 21.8 zeigt das von Programm *dreidim.c* angezeigte Bilder für die folgenden Werte:

```

Startwert für x: -2
Endwert für x: 4
Schrittweite für x (>0): 0.05
Startwert für y: -40
Endwert für y: 20
Schrittweite für y (>0): 0.5

```

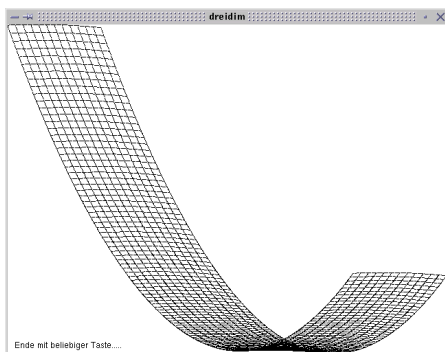
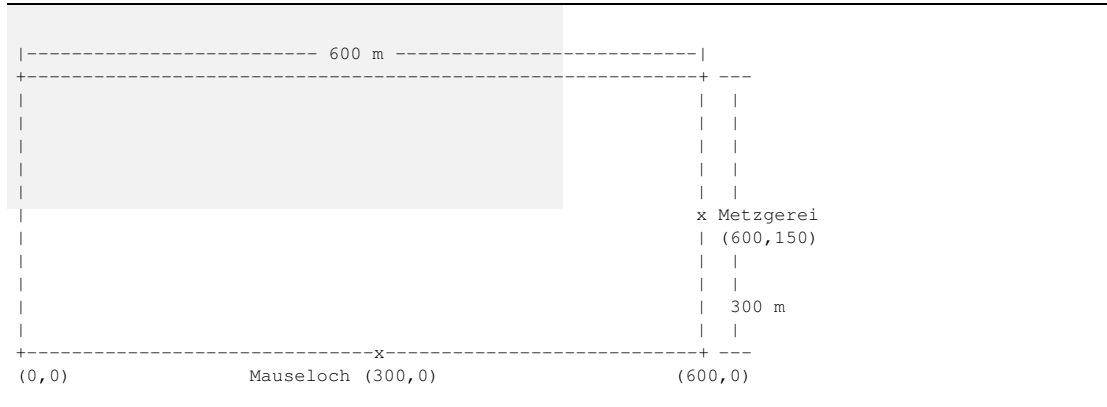


Abbildung 21.8: Dreidimensionaler Plot für die Funktion  $z = x^2 - y^2$

## 21.7 Katze jagt die Maus Physik(5;<110)

Eine Maus möchte einen Platz überqueren, der folgende Ausmaße hat:



Auf diesem Platz befindet sich jedoch nun immer eine Katze, welche die Maus, die aus dem ihr bekannten Mausloch kommt, gnadenlos verfolgt. Das Mausloch befindet sich an Position (300,0) im mathematischen Koordinatensystem. Während die Katze ständig ihre Richtung zur momentanen Mausposition hin ändert, läuft die Maus immer gerade auf die Metzgerei an Position (600,150) zu. Erstellen Sie ein C-Programm *katzmaus.c*, das diesen Verfolgungsprozeß graphisch am Bildschirm zeigt. Dieses Programm soll zunächst die Anfangsposition der Katze einlesen. Danach soll es die Geschwindigkeiten der Katze und der Maus (in m/s) einlesen. Zusätzlich soll der Benutzer festlegen können, in welchen Zeitintervallen die Katze ihre Richtung der momentanen Maus-Position anpaßt. Um eventuell die Verfolgungsjagd in Zeitlupe beobachten zu können, soll der Benutzer noch eine Verzögerung festlegen können, die zwischen jedem Verfolgungs-Schritt stattfindet. Einen möglichen Ablauf des Programms *katzmaus.c*, bei dem die Maus nicht überlebt, zeigen die Abbildungen 21.9 und 21.10, wobei hierfür die folgenden Eingaben gemacht wurden:

```
Startposition der Katze: x-Wert: 100; y-Wert: 250
Geschwindigkeit der Katze (in m/s): 9; Geschwindigkeit der Maus (in m/s): 6
Katze aendert ihre Richtung in welchen Zeitintervallen (Sek.): 0.1
Verzögerung bei jedem Schritt (in Millisekunden): 1
```

Einen möglichen Ablauf des Programms *katzmaus.c*, bei dem die Maus überlebt, zeigt Abbildung 21.11, wobei hierfür die folgenden Eingaben gemacht wurden:

```
Startposition der Katze:
x-Wert: 0; y-Wert: 200
Geschwindigkeit der Katze (in m/s): 7; Geschwindigkeit der Maus (in m/s): 5
Katze aendert ihre Richtung in welchen Zeitintervallen (Sek.): 0.1
Verzögerung bei jedem Schritt (in Millisekunden): 1
```

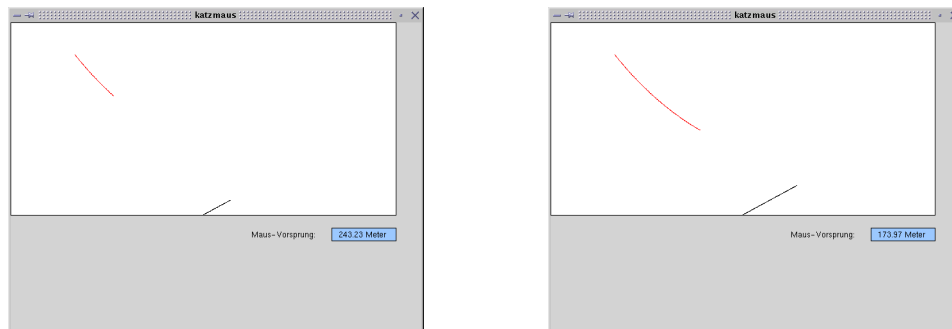


Abbildung 21.9: Katze jagt Maus (Schnappschuß am Anfang und etwas später)

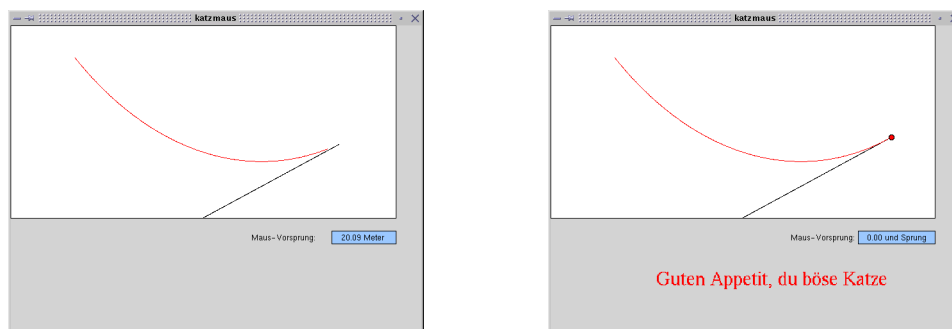


Abbildung 21.10: Katze jagt Maus (Schnappschuß kurz vor dem Ende und beim Einholen)

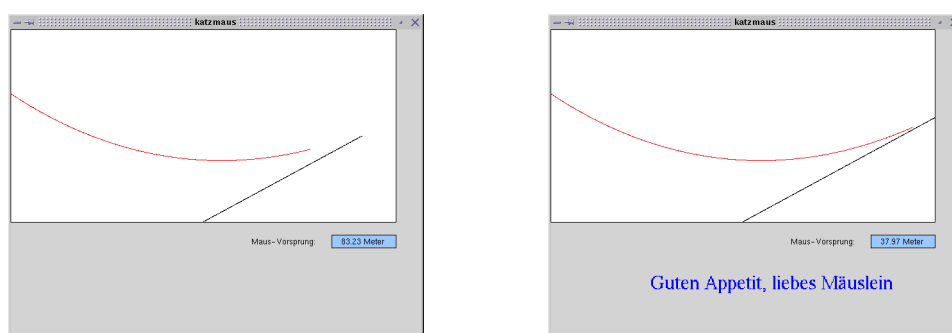


Abbildung 21.11: Katze jagt Maus (Schnappschuß kurz vor Ende und nachdem Maus entkommen)

## 21.8 Basketball spielen Physik(6;<150)

Erstellen Sie ein C-Programm *basket.c*, das zunächst einen Basketballkorb an einer zufällig ermittelten Position in der rechten Hälfte des Graphikfensters zeichnet und am linken unteren Rand des Graphikfensters einen Ball hüpfen lässt. Dieser Ball soll am Boden die Startposition einnehmen, wenn der Benutzer mit der Eingabe des Abwurfwinkels (siehe links Abbildung 21.12) beginnt. Nach der Eingabe dieses Winkels muss der Benutzer noch die Anfangsgeschwindigkeit, mit welcher der Ball abgeworfen werden soll, in Meter pro Sekunde eingeben (siehe rechts in Abbildung 21.12).

Nach diesen Eingaben soll die Flugbahn des Basketballs simuliert werden; siehe auch Abbildung 21.13. Wird der Korb getroffen, so soll dies dem Benutzer gemeldet werden und der Ball dann senkrecht zu Boden fallen; siehe auch Abbildung 21.14. Grundsätzlich gilt, dass beim Auftreffen des Balls auf dem Boden, dieser mit dem gleichen Winkel wieder abspringt, mit dem er aufsprang, allerdings nur mit halber Geschwindigkeit. Während des Flugs soll keinerlei Reibung (wie z.B. Luftwiderstand) berücksichtigt werden.



Abbildung 21.12: Eingabe des Abwurfwinkels und der Abwurfgeschwindigkeit

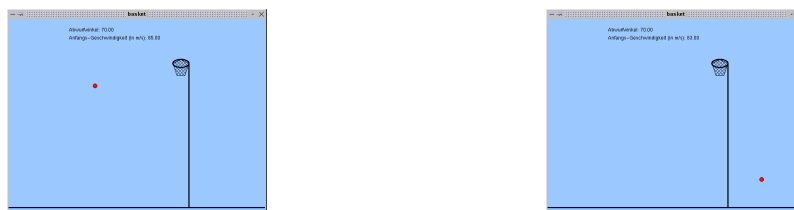


Abbildung 21.13: Simulation der Flugbahn des Basketballs

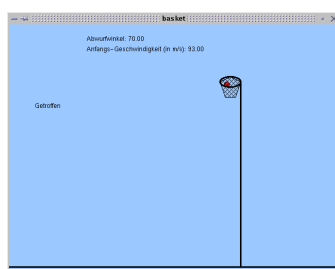


Abbildung 21.14: Anzeige bei einem Treffer im Basketballspiel



## 21.9 Spiralen über Kreise *Mathematik(5;<100)*

Auf dem Bildschirm sollen drei Kreise gegeneinander versetzt um den Mittelpunkt kreisen. Der Rotations-Radius und der Radius der Kreise soll dabei langsam größer werden. Die drei Kreise sollen dabei mit unterschiedlichen, aber gleichbleibenden Farben gezeichnet werden. Den x-, den y-Faktor und den Vergrößerungsfaktor der Rotation soll dabei der Benutzer ebenso eingeben können wie den Radius der rotierenden Kreise. Dieser Kreisradius soll sich dann durch das Programm *rotkreis.c* bei allen drei Kreisen langsam und gleichmäßig bei der Rotation vergrößern.

Abbildung 21.15 zeigt das Rotieren der Kreise für folgende Eingabedaten:

X-Faktor: 2; Y-Faktor: 2; Kreis-Radius: 2; Radius-Faktor: 2

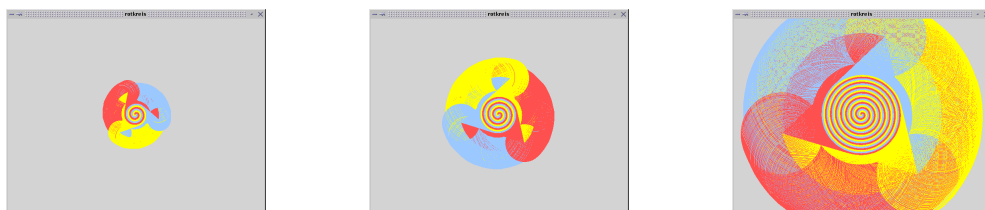


Abbildung 21.15: Rotieren von drei Kreisen

Abbildung 21.16 zeigt das Rotieren der Kreise für folgende Eingabedaten:

X-Faktor: 10; Y-Faktor: 2; Kreis-Radius: 3; Radius-Faktor: 4

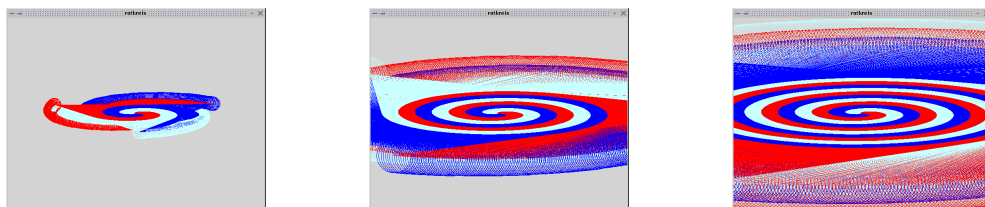


Abbildung 21.16: Rotieren von drei Kreisen

## 21.10 Kreis mit `putpixel()` zeichnen *Mathematik(4;<30)*

Erstellen Sie ein C-Programm *kreis.c*, das zunächst einen Radius einliest und dann einen Kreis mit diesem Radius malt. Dieser Kreis soll allerdings nicht unter Zuhilfenahme von `circle()`, sondern mit `putpixel()` gemalt werden.

## 21.11 Kreise mit Linien zeichnen *Mathematik(5;<100)*

Erstellen Sie ein C-Programm *linkreis.c*, das durch Ziehen von Linien zwischen den Bildschirmrändern einen Kreis im Mittelpunkt des Bildschirms erzeugt. Der Radius des Kreises soll dabei zufällig bestimmt werden. Auch sollen die Linien alle mit einer zufällig ermittelten gleichen Farbe gezeichnet werden. Nachdem ein Kreis komplett dargestellt ist, soll der Radius um einen zufälligen Wert erhöht werden und der ganze Vorgang für diesen neuen Kreis wiederholt werden. Dies soll solange wiederholt werden, bis der Radius größer als der halbe Bildschirm ist.

Mögliche durch das Programm *linkreis.c* gemalte Bilder zu unterschiedlichen Zeitpunkten sind in den Abbildungen 21.17 und 21.18 gezeigt.

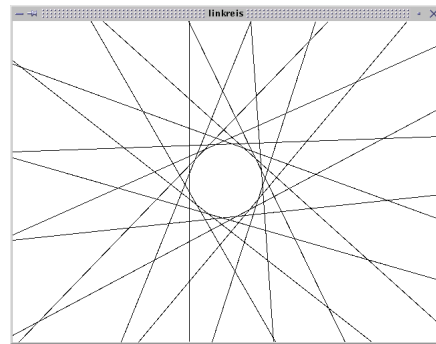
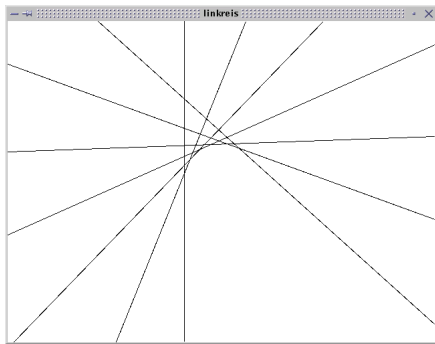


Abbildung 21.17: Kreise mit Linien zeichnen (ziemlich am Anfang)

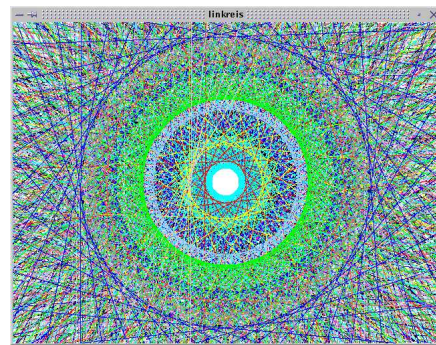
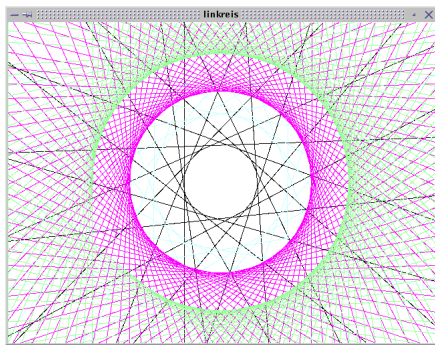


Abbildung 21.18: Kreise mit Linien zeichnen (später)

## 21.12 Der Ellipsenwurm *Mathematik(5;<70)*

Erstellen Sie ein C-Programm *wurmclip.c* bei dem eine gefüllte Ellipse von links oben beginnend über den Bildschirm wandert. Der x- und der y-Radius dieser Ellipse soll sich ständig ändern. Das Programm soll beim Drücken der **(ESC)**-Taste beendet werden. Drückt der Benutzer die Taste **(P)**, so soll der Bildschirm gelöscht werden und eine neue Ellipse vom linken oberen Bildschirmrand aus zu wandern beginnen. Beim Drücken jeder anderen Taste soll das Programm nur anhalten und erst auf einen weiteren Tastendruck hin die Ellipse weiter wandern lassen.

Mögliche durch das Programm *wurmclip.c* gemalte Bilder zu unterschiedlichen Zeitpunkten sind in den Abbildungen 21.19 und 21.20 gezeigt.

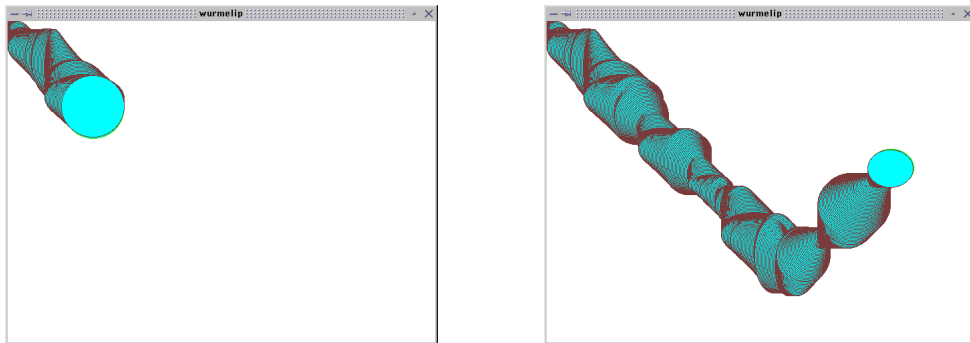


Abbildung 21.19: Der Ellipsenwurm (am Anfang)

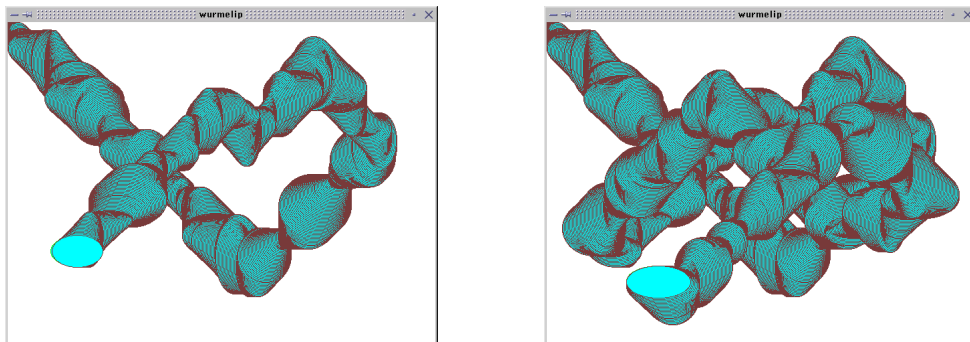


Abbildung 21.20: Der Ellipsenwurm (später)

## 21.13 Huhn im dreieckigen Kornfeld *Fraktale(4;<120)*

Ein Huhn befindet sich auf einem dreieckigen abgeernteten Kornfeld. Es pickt nun die dort überall herumliegenden Körner nach folgender Methode auf:

Es wählt sich zufällig eine der drei Ecken aus und läuft gerade auf diese Ecke zu. Nach der Hälfte des Weges bleibt es stehen und pickt dort ein Korn auf. Nun sucht es sich wieder zufällig einen der drei Eckpunkte aus, marschiert in Richtung dieses Punkts, bleibt aber auf der Hälfte der Strecke stehen und pickt das Korn auf usw.

Erstellen Sie ein C-Programm *huhn.c*, das dem Benutzer zunächst die drei Eckpunkte des dreieckigen Kornfelds durch drei Klicks mit der linken Maustaste festlegen lässt. Danach muss der Benutzer noch eingeben, wie oft das Huhn picken soll, bevor eine Simulation beginnt. Der Startpunkt des Huhns soll dabei zufällig gewählt werden.

Klickt der Benutzer während oder am Ende eines Simulationsvorgangs auf die mittlere Maustaste, wird ein neuer Simulationsvorgang gestartet, wobei er mit der linken Maustaste wieder die Eckpunkte des dreieckigen Kornfelds festlegen und die Anzahl der Picks eingeben muss. Bei jedem Simulationsvorgang soll Ihr Programm die Pickpunkte des Huhns durch Ausgabe eines farbigen Pixels an dieser Stelle kennzeichnen, wobei bei jedem Simulationsvorgang eine zufällig gewählte Farbe zu verwenden ist.

Klickt der Benutzer während oder am Ende eines Simulationsvorgangs auf die rechte Maustaste, wird das Programm beendet.

Lassen Sie sich überraschen. Die Ausgabe ergibt eine nicht erwartete fraktale Struktur: das sogenannte Sierpinski-Sieb. Dieses Sieb ist nach seinem Entdecker, dem polnischen Mathematiker *Waclaw Sierpinski* benannt.

## 21.14 Die Kochsche Schneeflocke *Fraktale(6;<100)*

Erstellen Sie ein C-Programm *koch.c*, das die Kochsche Schneeflocke auf den Bildschirm malt. Der Benutzer soll dabei vor dem Malen eingeben können, ob

- ☐ Farbe zu verwenden ist,
- ☐ bei den Transformationen das vorherige Bild zu löschen ist und
- ☐ die Kurve ausgefüllt oder nicht ausgefüllt zu zeichnen ist.

Die Abbildungen 21.21 und 21.22 zeigen die ersten Folge von eingeblendeten Kochkurven, wobei die nächste Transformation immer durch einen Tastendruck ausgelöst wird. Der Schwierigkeitsgrad dieses Programms ist sehr hoch, weshalb Ihnen vielleicht der folgende kleine Tipp weiterhilft: *Das Zeichnen der Linien kann unter Zuhilfenahme des 4er-Systems erfolgen.*

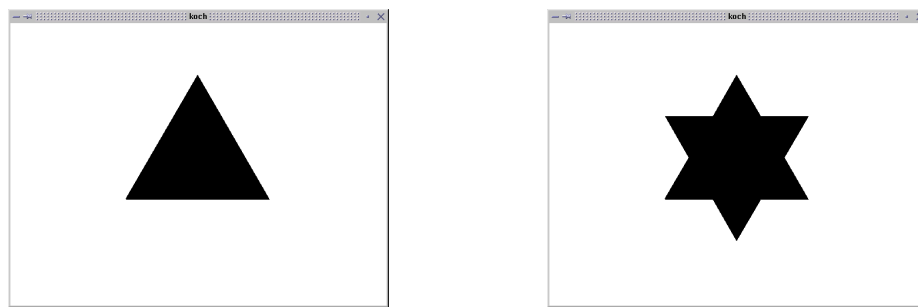


Abbildung 21.21: Ausgangspunkt und erste Transformation

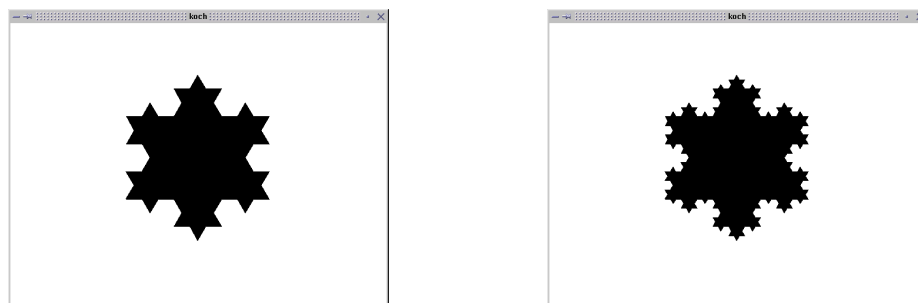


Abbildung 21.22: Zweite und dritte Transformation

## 21.15 Die Mandelbrot-Menge *Chaos(6;<250)*

Die Mandelbrot-Menge ist eine Menge von Punkten. Jeder Punkt auf der komplexen Ebene – also jede komplexe Zahl – gehört entweder zur Menge oder nicht. Man kann die Menge definieren, indem man für jeden Punkt der Ebene Tests durchführt. Es geht dabei um eine einfache iterative Arithmetik.

Um einen Punkt zu überprüfen, nimmt man die entsprechende komplexe Zahl, erhebt sie ins Quadrat, addiert die ursprüngliche Zahl, quadriert wieder das Ergebnis, addiert wieder die ursprüngliche Zahl, quadriert das Ergebnis – und so weiter immer nach demselben Schema. Wenn die Gesamtsumme gegen unendlich läuft, gehört der Punkt nicht zur Mandelbrot-Menge. Bleibt die Summe endlich, gehört der Punkt zur Mandelbrot-Menge.

### Allgemeine Hinweise zu komplexen Zahlen:

Eine komplexe Zahl setzt sich immer aus zwei Teilen zusammen: dem Realteil (x-Achse) und dem Imaginärteil (y-Achse):

$$kzahl = x + iy; \quad i = \sqrt{-1}$$

Zur Addition zweier komplexer Zahlen gilt folgende Formel:

$$\begin{aligned} (a + ib) + (c + id) &= (a + c) + (b + d)i \\ \text{neuer Realteil :} & \quad a + c \\ \text{neuer Imaginärteil :} & \quad b + d \end{aligned}$$

Zur Multiplikation zweier komplexer Zahlen gilt folgende Formel:

$$\begin{aligned} (a + ib) \cdot (c + id) &= a \cdot b + a \cdot id + ib \cdot c + i \cdot i \cdot b \cdot d & (i^2 = -1) \\ &= a \cdot b - b \cdot d + (a \cdot d + b \cdot c)i \\ \text{neuer Realteil :} & \quad a \cdot b - b \cdot d \\ \text{neuer Imaginärteil :} & \quad a \cdot d + b \cdot c \end{aligned}$$

Da hier die Zahl mit sich selbst multipliziert wird, gilt folgende Formel:

$$\begin{aligned} (a + ib) \cdot (a + ib) &= a \cdot a + 2 \cdot a \cdot ib + i \cdot i \cdot b \cdot b & (i^2 = -1) \\ &= a \cdot a - b \cdot b + (2 \cdot a \cdot b)i \\ \text{neuer Realteil :} & \quad a^2 - b^2 \\ \text{neuer Imaginärteil :} & \quad 2 \cdot a \cdot b \end{aligned}$$

Erstellen Sie ein C-Programm *mandel.c*, das zunächst den Bereich der reellen Komponenten und dann den Bereich der komplexen Komponenten einliest. Danach soll dieses Programm die maximale Anzahl von Iterationen und einen Schwellwert einlesen. Schließlich soll Benutzer noch die Bildgröße festlegen können.

Für jeden einzelnen Punkt in der komplexen Ebene wird dann solange iteriert, bis entweder die maximal vorgegebene Zahl von Iterationen erreicht ist, oder aber der Schwellwert ( $a^2 + b^2 > \text{Schwellwert}$ ) überschritten ist. Abhängig von der Iterationszahl wird dann die Farbe des jeweiligen Pixels in der simulierten komplexen Ebene bestimmt.

Nachdem ein Bild gezeichnet ist, soll der Benutzer mittels der Maus sich einen Teilbereich herauszoomen lassen können. Dazu muss der Benutzer bei gedrückter linker Maustaste seinen Bereich eingrenzen, wobei das Programm ihm mit einem gestrichelten Rechteck den aktuell

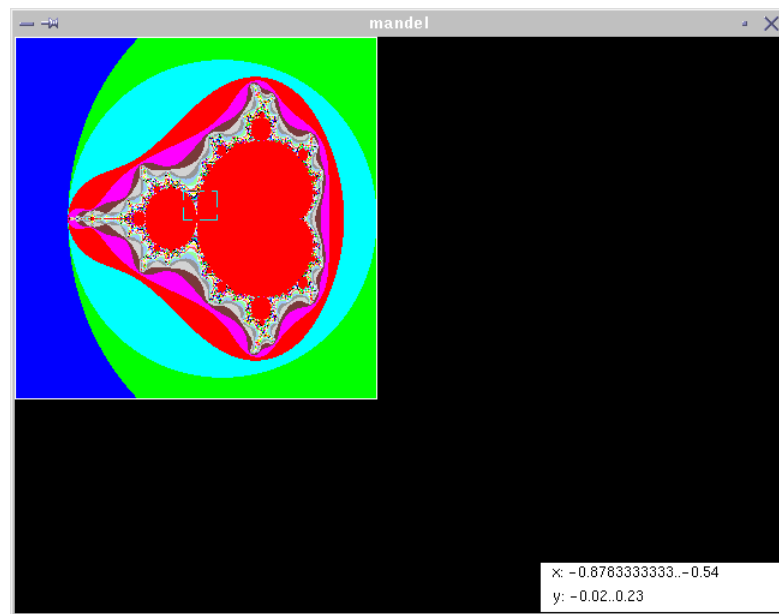


Abbildung 21.23: Mandelbrot für reel: -2.5..1; imaginär: -1.5..1.5; max. Iterationen: 100; Schwellwert: 4; Bildgröße 300x300

gewählten Bereich anzeigt, wie dies auch in Abbildung 21.23 erkennbar ist. Erst wenn der Benutzer die Maustaste losläßt, wird der ausgewählte Bereich herausgezoomt; siehe auch Abbildungen 21.24, 21.25 und 21.26. Weitere interessante Bilder ergeben sich z.B. bei den folgenden Daten:

#### **Spirale**

x: [-0.7666, -0.7654]; y: [0.10021, 0.10151]; Iterationen: 500

#### **Seitenarm**

x: [-0.74825, -0.7425]; y: [0.0962, 0.10067]; Iterationen: 500

#### **Schmetterling**

x: [-0.76498, -0.76461]; y: [0.10056, 0.10082]; Iterationen: 500

#### **Trichter**

x: [-0.74758, -0.74624]; y: [0.10671, 0.10779]; Iterationen: 1000

#### **Seepferde**

x: [-0.745538, -0.745054]; y: [0.112881, 0.113236]; Iterationen: 1000

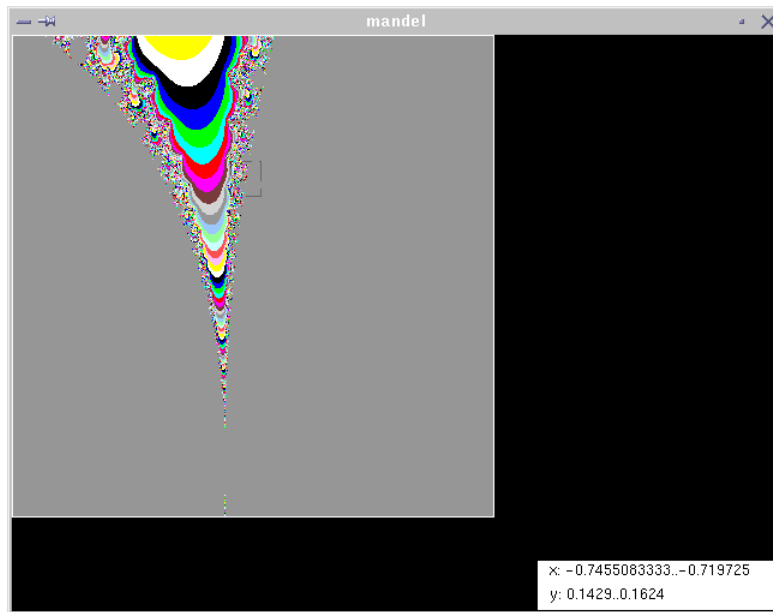


Abbildung 21.24: Mandelbrot für ausgewählten Bereich in Abbildung 21.23: Iterationen: 200; Schwellwert: 30; Bildgröße 400x400

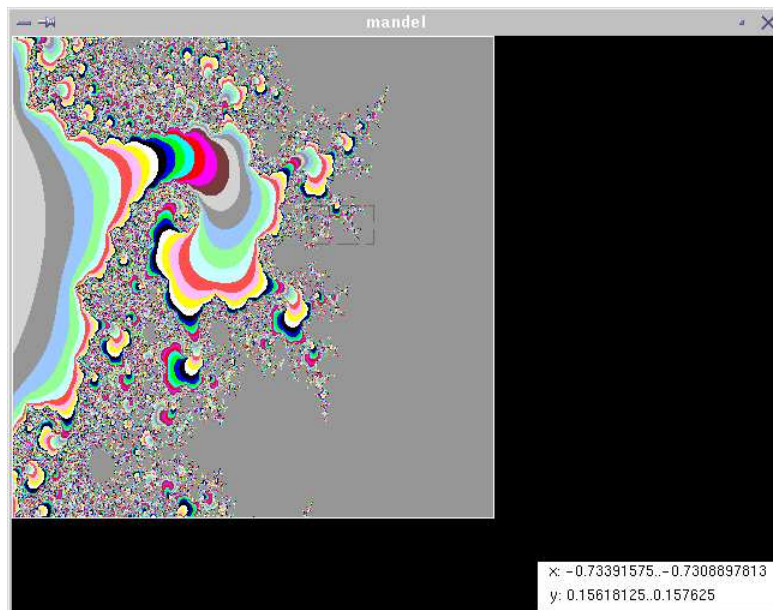


Abbildung 21.25: Mandelbrot für ausgewählten Bereich in Abbildung 21.24: Iterationen: 200; Schwellwert: 30; Bildgröße 400x400



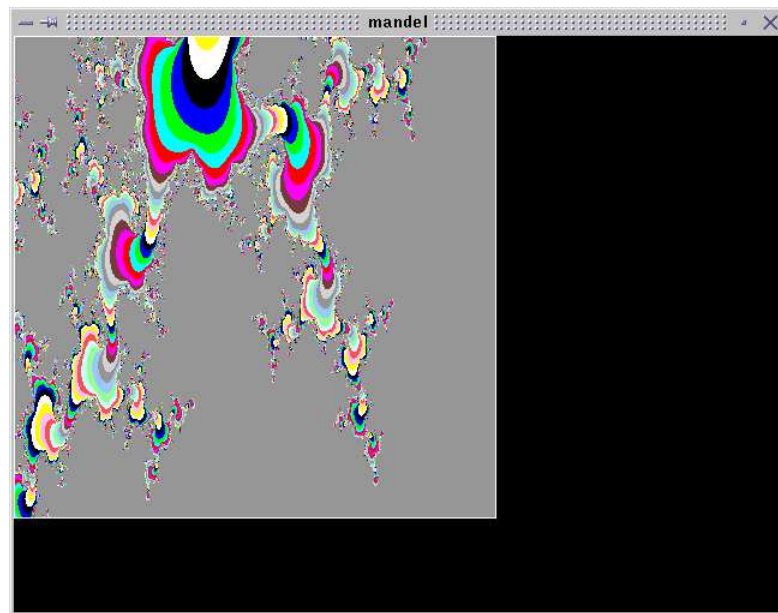


Abbildung 21.26: Mandelbrot für ausgewählten Bereich in Abbildung 21.25: Iterationen: 200; Schwellwert: 30; Bildgröße 400x400

## 21.16 Ein Malprogramm *Zeichnen(6;<250)*

Erstellen Sie ein C-Programm *male.c*, mit dem Sie in einem eingeblendeten Fenster freihändig mit der Maus malen können. Dabei sollen folgende Einstellungen möglich sein:

- ☐ Farbe
- ☐ Pinselstärke
- ☐ Pinselform (Kreis oder Rechteck)

Das eingeblendete Fenster soll dabei in 5 verschiedene Unterfenster aufgeteilt sein:

### 1. Malfenster

ist nahezu das ganze Fenster, bis auf die Einstellungsleiste unten. Drückt der Benutzer hier die linke Maustaste und bewegt dabei die Maus, so werden beim Bewegen ständig mit der aktuellen Malfarbe und der Pinselstärke in der aktuellen Pinselform (ausgefüllte Kreise oder ausgefüllte Rechtecke) an die momentane Cursorposition gemalt. Das Loslassen der linken Maustaste entspricht dem Abheben des Pinsels vom Bildschirm. Der Benutzer kann dann eventuell neu positionieren und an einer anderen Stelle im Malfenster wieder mit dem Malen beginnen. Drückt der Benutzer die mittlere Maustaste, dann wird das ganze Malfenster gelöscht. Drückt er die rechte Maustaste, so wird in das Farbfenster umgeschaltet.

### 2. Farbfenster

Dieses Fenster befindet sich in der unteren Leiste. Hat der Benutzer hierher vom Malfenster (mit dem Drücken der rechten Maustaste) umgeschaltet, so kann er hier durch wiederholtes Drücken der linken Maustaste eine neue Malfarbe einstellen. Drückt er die rechte Maustaste, so wird in das Pinselstärke-Fenster umgeschaltet.

### 3. Pinselstärke-Fenster

Dieses Fenster befindet sich auch in der unteren Leiste. Hat der Benutzer hierher vom Farbfenster (mit dem Drücken der rechten Maustaste) umgeschaltet, so kann er hier durch wiederholtes Drücken der linken Maustaste die Pinselstärke hochsetzen. Erreicht die Pinselstärke ihre oberste Grenze, wird sie wieder zurückgesetzt. Drückt der Benutzer hier die rechte Maustaste, so wird in das Pinselform-Fenster umgeschaltet.

### 4. Pinselform-Fenster

Dieses Fenster befindet sich auch in der unteren Leiste. Hat der Benutzer hierher vom Pinselstärke-Fenster (mit dem Drücken der rechten Maustaste) umgeschaltet, so kann er hier durch Drücken der linken Maustaste die Pinselform von Kreis auf Rechteck und umgekehrt umstellen. Drückt der Benutzer hier die rechte Maustaste, so wird wieder in das Malfenster umgeschaltet.

### 5. Koordinaten-Fenster

Hier werden ständig die aktuellen Maus-Koordinaten angezeigt, wenn der Benutzer sich im Malfenster befindet.

Das Programm wird beendet, wenn eine Eingabe von der Tastatur erfolgt.

Zunächst befindet sich der Benutzer im Malfenster, in dem er nun beliebig malen kann, wie dies in Abbildung 21.27 gezeigt ist.

Nach dem Verstellen der Farbe, Pinselstärke und Pinselform, und einem erneuten Malen könnte sich z.B. das in Abbildung 21.28 gezeigte Bild ergeben.

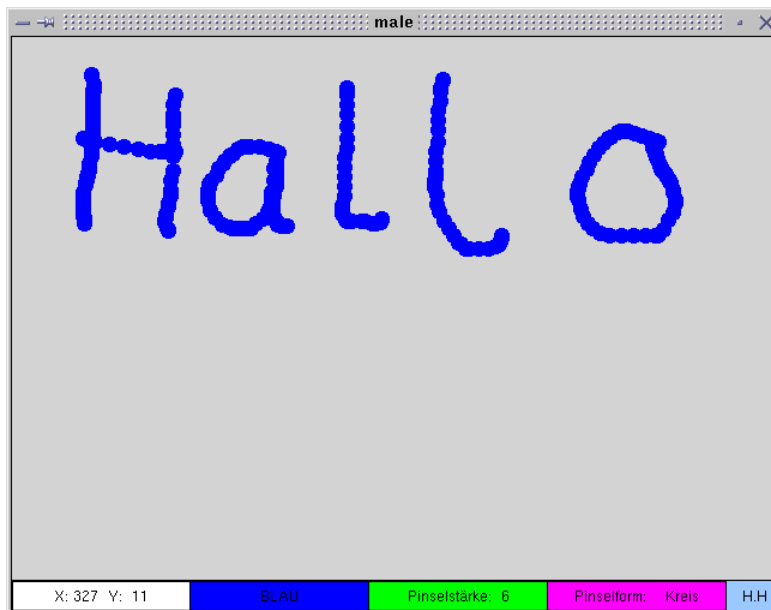


Abbildung 21.27: Malen mit der Maus im Malfenster

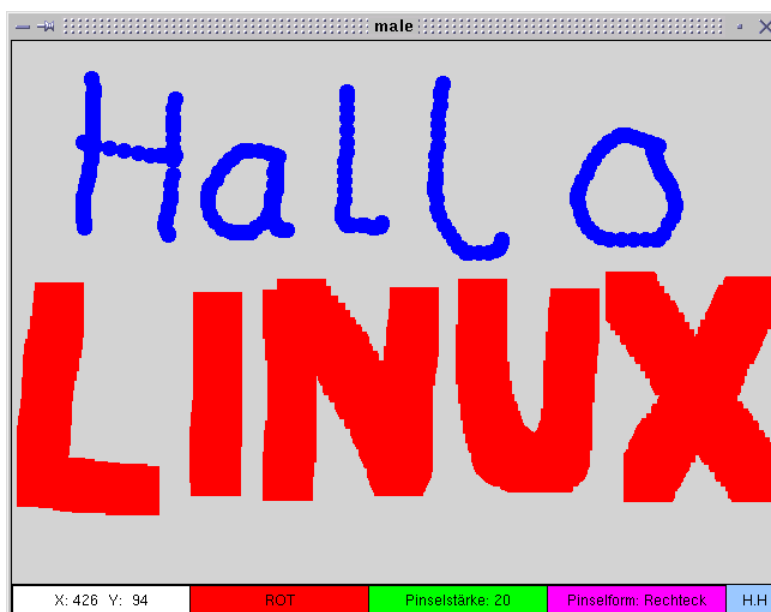


Abbildung 21.28: Malen nach dem Ändern der Farbe, Pinselstärke und Pinselform

# 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 !!!!!

```

## 22.4 Ellipsen-Prototypen

### 22.4.1 Maximum aus vielen Zahlen *Allgemein(3;<40)*

Schreiben Sie eine Funktion `vielmax()`, die das Maximum beliebig vieler Ganzzahlen ermittelt. Zum Testen der Funktion rufen Sie diese in `main()` (in Ihrem Programm `vielmax.c`) mit unterschiedlich vielen Argumenten auf. Möglicher Ablauf des Programms `vielmax.c`:

```
Testprogramm fuer Funktion vielmax()
=====

Das Maximum der Zahlen 12, 17, 3, 6, 24, 8 ist: 24
Das Maximum der Zahlen 105, 77, 3, 54 ist: 105
```

## 22.5 Rekursive Funktionen

### 22.5.1 Binomialkoeffizient *Mathematik(3;<50)*

Der Binomialkoeffizient lässt sich für  $0 \leq k \leq n$  auch rekursiv definieren:

$$\binom{n}{k} = \begin{cases} 1 & : \text{wenn } k = 0 \text{ oder } k = n \\ \binom{n-1}{k} + \binom{n-1}{k-1} & : \text{sonst} \end{cases}$$

Erstellen Sie ein Programm `binom.c`, das Binomialkoeffizienten entsprechend dieser Formel rekursiv berechnet. Mögliche Abläufe des Programms `binom.c` sind:

```
Wieviele Positionen: 49 (↩)
Wieviele Elemente (muss <= 49 sein): 6 (↩)

/   49 \
|       | = 13983816
\   6   /
```

```
Wieviele Positionen: 124 (↩)
Wieviele Elemente (muss <= 124 sein): 3 (↩)

/   124 \
|       | = 310124
\   3   /
```

```
Wieviele Positionen: 2456 (↩)
Wieviele Elemente (muss <= 2456 sein): 1 (↩)

/  2456 \
|       | = 2456
\   1   /
```



### 22.5.2 Größter gemeinsamer Teiler *Mathematik(4;<40)*

Der größte gemeinsame Teiler (ggT) zweier Zahlen lässt sich auch rekursiv definieren:

$$\text{ggT}(n, m) = \begin{cases} n & : \text{wenn } m = 0 \\ \text{ggT}(m, n \bmod m) & : \text{sonst} \end{cases}$$

Erstellen Sie ein C-Programm *ggtrekur.c*, das den größten gemeinsamen Teiler von mehreren Zahlen unter Verwendung dieser rekursiven Formel ermittelt.

Mögliche Abläufe des Programms *ggtrekur.c* sind:

Gib nicht-negative ganze Zahlen ein (Ende=0)

```
1. Zahl: 818127 (↩)
2. Zahl: 999999 (↩)
3. Zahl: 1071 (↩)
4. Zahl: 0 (↩)
====> ggt = 9
```

```
1. Zahl: 2376 (↩)
2. Zahl: 3751 (↩)
3. Zahl: 23122 (↩)
4. Zahl: 0 (↩)
====> ggt = 11
```

```
1. Zahl: 53536 (↩)
2. Zahl: 78283 (↩)
3. Zahl: 89372 (↩)
4. Zahl: 0 (↩)
====> ggt = 1
```

### 22.5.3 Umwandeln einer Dezimalzahl in Dualzahl *DV-Wissen(3;<30)*

Ein Algorithmus zur Umwandlung einer Dezimalzahl in eine Dualzahl ist:

```
Solange zahl ungleich 0 führe folgende Schritte durch:
    rest = zahl % 2 (Rest von Division durch 2)
    zahl = zahl / 2
Gib errechnete Reste in umgekehrter Reihenfolge (von unten nach oben) aus.
```

```
Beispiel:    12 : 2 = 6 Rest 0 ^
             6 : 2 = 3 Rest 0 |
             3 : 2 = 1 Rest 1 | Ausgabe in umgekehrter Reihenfolge
             1 : 2 = 0 Rest 1 | Dualzahl zu 12 ist somit: 1100
```


Erstellen Sie ein C-Programm *dualwand.c*, das diese Umwandlung rekursiv löst.


```
Gib eine Dezimalzahl ein: -1 (↩)
11111111111111111111111111111111
```

```
Gib eine Dezimalzahl ein: 12345 (↩)
11000000111001
```

### 22.5.4 Quersumme und Umdrehen einer Zahl *Mathematik(3;<40)*

Erstellen Sie ein C-Programm *quersum.c*, das eine Zahl einliest und dann zu dieser Zahl unter Verwendung von Rekursion die Quersumme berechnet, bevor es diese Zahl dann ebenfalls unter Verwendung von Rekursion rückwärts ausgibt.

```
Gib eine Zahl ein: 12345   
----Quersumme: 15  
----umgedreht: 54321
```

```
Gib eine Zahl ein: 418223789   
----Quersumme: 44  
----umgedreht: 987322814
```

### 22.5.5 Das Terrassenproblem *Allgemein(4;<30)*

Eine rechteckige Terasse der ganzzahligen Länge  $a$  und Breite  $b$  soll mit möglichst großen quadratischen Platten ausgelegt werden. Welche Kantenlänge müssen die Platten haben?

Erstellen Sie ein C-Programm *terasse.c*, das diese Aufgabenstellung löst.

Mögliche Abläufe des Programms *terasse.c*:

```
Auslegen einer Terasse mit moeglichst grossen quadrat. Steinen
=====
```

```
Breite der Terasse: 270 (←)
Laenge der Terasse: 198 (←)
```

```
----> Steine mit 18 x 18
```

```
Auslegen einer Terasse mit moeglichst grossen quadrat. Steinen
=====
```

```
Breite der Terasse: 100 (←)
Laenge der Terasse: 50 (←)
```

```
----> Steine mit 50 x 50
```

```
Auslegen einer Terasse mit moeglichst grossen quadrat. Steinen
=====
```

```
Breite der Terasse: 1001 (←)
Laenge der Terasse: 990 (←)
```

```
----> Steine mit 11 x 11
```

```
Auslegen einer Terasse mit moeglichst grossen quadrat. Steinen
=====
```

```
Breite der Terasse: 99019 (←)
Laenge der Terasse: 9291 (←)
```

```
----> Steine mit 1 x 1
```

## 22.6 Zeiger auf Funktionen

### 22.6.1 Der Baum des Pythagoras *Allgemein(6;<100)*

Der *Baum des Pythagoras* setzt sich aus lauter Quadraten zusammen, die so um rechtwinklige Dreiecke angeordnet sind, dass sie den Satz des Pythagoras illustrieren. Das Seitenverhältnis der beiden Katheten zueinander bestimmt dabei die "Schiefe des Baums". In Abbildung 22.1 haben die beiden Katheten das Seitenverhältnis 4:5 (entspricht 0.8).

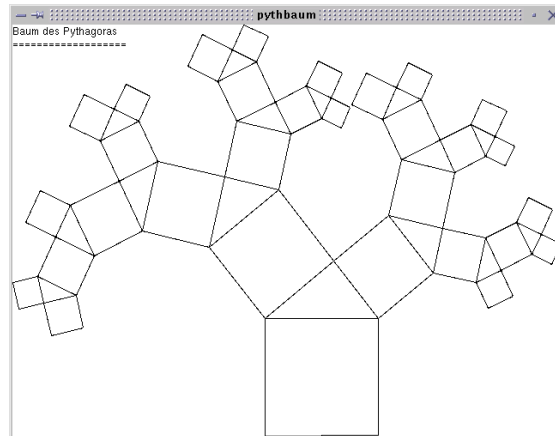


Abbildung 22.1: Der Baum des Pythagoras (Seitenverhältnis 4:5)

Erstellen Sie nun ein C-Programm *pythbaum.c*, das zunächst das Seitenverhältnis und die maximale Länge der kleinsten Linie (in Pixel) einliest, bevor es dann den Baum des Pythagoras genau auf den Bildschirm passend ausgibt. Je kleiner die maximale Länge der kleinsten Linie (in Pixel) gewählt wird, um so mehr verästelt sich dieser Baum. Die Abbildungen 22.2 bis 22.5 zeigen solche vom Programm *pythbaum.c* eingeblendete Pythagorasbäume für unterschiedliche Seitenverhältnisse und maximale Längen (für die kleinsten Linien).

#### Lösungshilfe:

Zur Lösung muss die geometrische Gesetzmäßigkeit des Baumes erkannt werden. Die Figur soll den Satz des Pythagoras veranschaulichen: Das Quadrat über der Hypotenuse eines rechtwinkligen Dreiecks ist gleich der Summe der Quadrate über die Katheten:  $a^2 + b^2 = c^2$ .

Der rekursive Algorithmus lautet dann:

Zeichne über eine Grundseite eine Quadrat. Betrachte die dieser gegenüberliegende Quadratseite als Hypotenuse (c) eines rechtwinkligen Dreiecks. Zeichne, solange die Auflösung es zulässt, die Quadrate über der Ankathete (a) und der Gegenkathete (b) dieses Dreiecks nach dem gleichen Algorithmus.

Man kann sich hier überlegen, dass der ganze pythagoräische Baum in einem Zug ohne Absetzen des Zeichenstifts als sogenannter *Eulerscher Zyklus* erzeugt werden kann, wie dies in Abbildung 22.6 gezeigt ist.

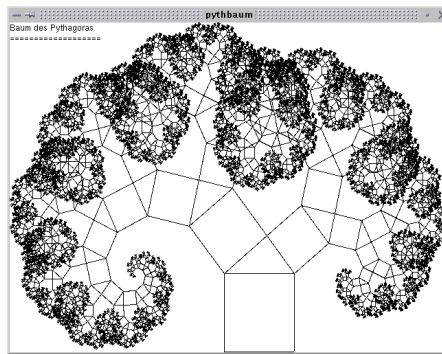


Abbildung 22.2: Baum des Pythagoras (Seitenverhältnis: 0.8; kleinste Linie: 2 Pixel)

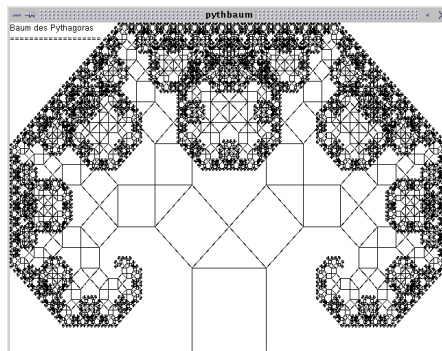


Abbildung 22.3: Baum des Pythagoras (Seitenverhältnis: 1; kleinste Linie: 2 Pixel)

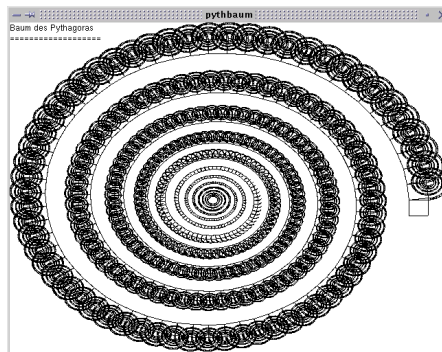


Abbildung 22.4: Baum des Pythagoras (Seitenverhältnis: 0.1; kleinste Linie: 2 Pixel)

Die Grundseite wird beim Ankathetenquadrat nachher und beim Gegenkathetenquadrat vorher gezeichnet. In der Graphentheorie heißt ein solcher geschlossener Kantenzug, bei dem jede Kante einmal und nur einmal durchlaufen wird, ein Eulerscher Zyklus.

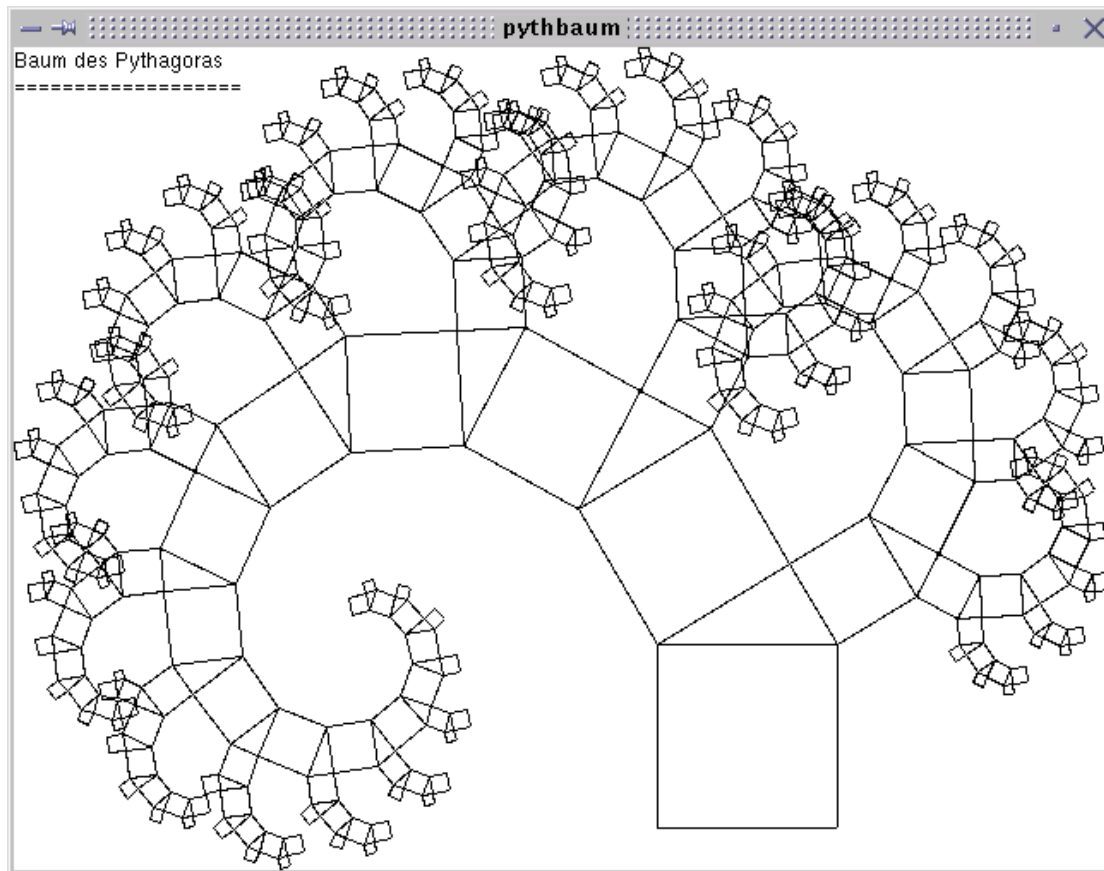


Abbildung 22.5: Baum des Pythagoras (Seitenverhältnis: 0.6; kleinste Linie: 10 Pixel)

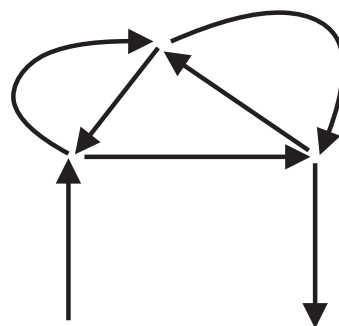


Abbildung 22.6: Der Euler Zyklus



# Kapitel 23

## Speicherklassen und Modultechnik

### Hinweis zu den Aufgaben aus Kapitel 23

Bei den Aufgaben aus den Kapiteln 23.1, 23.2 und 23.3 sollten die Fragestellungen zunächst durch eigene Überlegungen beantwortet und bei der Abnahme vorgebracht und begründet werden. Zur Überprüfung der Richtigkeit können diese danach z. B. abgeschrieben bzw. durch Paste-and-Copy zum Ablauf gebracht werden.

### 23.1 Gültigkeitsbereich, Lebensdauer, Speicherort

#### 23.1.1 Ausgabe des Programms `block.c` C–Syntax(3;<10)

Was gibt das folgende Programm `block.c` aus?

```
#include <stdio.h>

int i=0;

int main(void)
{
    int i=1;

    printf("i=%d\n", i);
    {
        int i=2;
        printf("i=%d\n", i);
        {
            i++;
            printf("i=%d\n", i);
        }
        printf("i=%d\n", i);
    }
    printf("i=%d\n", i);
    return(0);
}
```



## 23.2 Schlüsselwörter extern, auto, static und register

### 23.2.1 Ausgabe des Programms speikla1.c *C-Syntax(3;<10)*

Was gibt das folgende Programm *speikla1.c* aus?

```
#include <stdio.h>

int i=0;

int setzel(int x);
int setze2(int x);

int main(void)
{
    auto int i=5;
    setzel(i/2);    printf("i = %d\n", i);
    setzel(i=i/2);  printf("i = %d\n", i);
    i = setzel(i/2); printf("i = %d\n", i);
    setze2(i);      printf("i = %d\n", i);
    return(0);
}

int setzel(int i)
{
    i = i<=2 ? 5 : 0;
    return(i);
}

int setze2(int i)
{
    i = i%i * (i*i/(2*i)+4);    printf("i = %d\n", i);
    return(i);
}
```

### 23.2.2 Ausgabe des Programms `speikla2.c` C-Syntax(4;<10)

Was gibt das folgende Programm *speikla2.c* aus?

```
#include <stdio.h>

int i=1;

int setze(void);
int naechst(int x);
int letzt(int x);
int neu(int x);

int main(void)
{
    auto int i, j;
    i = setze();
    for (j=1; j<3; j++) {
        printf("i=%d, j=%d\n", i, j);
        printf("    naechst(i)=%d\n", naechst(i));
        printf("    letzt(i)=%d\n", letzt(i));
        printf("    neu(i+j)=%d\n", neu(i+j));
    }
    return(0);
}

int setze(void)
{
    return(i);
}

int naechst(int j)
{
    return(j=i++);
}

int letzt(int j)
{
    static int i=10;
    return(j=i--);
}

int neu(int i)
{
    auto int j=10;
    return(i=j+=i);
}
```

### 23.2.3 Ausgabe des Programms `speikla3.c` *C-Syntax*(5;<10)

Was gibt das folgende Programm aus, das sich aus den drei Modulen *speikla3.c*, *modulb.c* und *modulc.c* zusammensetzt?

Modul *speikla3.c*:

```
#include <stdio.h>

int i=1;

extern int setze(void);
extern int naechst(void);
extern int letzt(void);
extern int neu(int x);

int main(void)
{
    auto int i, j;

    i = setze();
    for (j=1; j<3; j++) {
        printf("i=%d, j=%d\n", i, j);
        printf("    naechst()=%d\n", naechst());
        printf("    letzt()=%d\n", letzt());
        printf("    neu(i+j)=%d\n", neu(i+j));
    }
    return(0);
}
```

Modul *modulb.c*:

```
static int i=10;

int naechst(void) { return(i+=1); }
int letzt(void)   { return(i-=1); }

int neu(int i)
{
    static int j=5;
    return(i=j+=i);
}
```

Modul *modulc.c*:

```
extern int i;

int setze(void)
{
    return(i);
}
```

## 23.3 Schlüsselwörter const und volatile

### 23.3.1 Konstante Zeiger und Zeiger auf Konstanten *C–Syntax(4;<10)*

Welche Anweisungen im folgenden Programm *constzei.c* sind erlaubt und welche nicht?

```
int main(void) {
    const double pi=3.14;
    double *dz;
    int var=100;
    int *const cz=&var;
    const int* zc=&var;
    const int* const czc=&var;

    pi = 6.2;
    dz = &pi;
    *dz = 6.2;

    *cz = 50;
    cz = zc;
    *zc = 500;
    zc = cz;
    *czc = 5000;
    czc = zc;
    return 0;
}
```

### 23.3.2 const-Parameter bei Funktionsdefinitionen *C–Syntax(4;<10)*

Welche Anweisung im folgenden Programm *consfunk.c* ist nicht erlaubt und was würde dieses Programm ausgeben, wenn diese Anweisung entfernt wird?

```
double wurz2(const double *arg);
double wurz3(double *arg);

int main(void) {
    double z1=10.0, z2;
    const double z3=8.0;

    z2 = wurz2(&z3); printf("%f\n", z2);
    z2 = wurz2(&z1); printf("%f\n", z2);
    z2 = wurz3(&z3); printf("%f\n", z2);
    z2 = wurz3(&z1); printf("%f\n", z2);
    printf("%f\n%f\n", z1, z3);
    return(0);
}

double wurz2(const double *arg) {
    *arg = 27.0;
    return(pow(*arg,0.5));
}

double wurz3(double *arg) {
    *arg = 32.0;
    return(pow(*arg,1/3.0));
}
```

## 23.4 Die Modultechnik

### 23.4.1 Turingmaschine zur binären Addition von 1 *Informatik(5;<200)*

Erstellen Sie unter Verwendung der Modultechnik ein Programm (*dualaddi.c, adturing.c, fehler.c, bildschi.c*), das eine duale Addition von 1 durchführt.

So soll z.B. diese Turingmaschine aus der Bandinschrift

1100#

die Bandschrift

1101#

produzieren, oder aus der Bandinschrift

11011#

die Bandinschrift

11100#

produzieren.

### 23.4.2 Verdopplungs-Turingmaschine *Informatik(5;<220)*

Erstellen Sie unter Verwendung der Modultechnik ein Programm (*verdoppl.c, doturing.c, fehler.c, bildschi.c*), das eine Verdopplung der auf dem Band angegebenen Striche durchführt. Wenn  $x$  Striche auf dem Band angegeben sind, so soll das Band danach  $2 \cdot x$  Striche enthalten.

So soll z.B. diese Verdopplungs-Turingmaschine aus der Bandinschrift

III#

die Bandschrift

####IIIIII

produzieren.

### 23.4.3 Multiplizier-Turingmaschine *Informatik(6;<300)*

Erstellen Sie unter Verwendung der Modultechnik ein Programm (*multipli.c, muturing.c, fehler.c, bildschi.c*), das eine Turingmaschine simuliert, die zwei Zahlen multipliziert.

So könnte z.B. diese Multiplizier-Turingmaschine aus der Bandinschrift

III#IIII#

die Bandschrift

AAA#IIII#IIIIIIIIIIII

produzieren.

# 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.

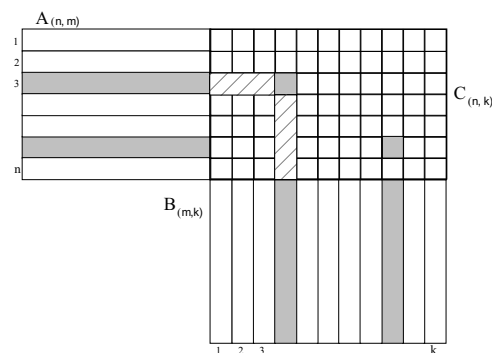


Abbildung 25.1: Multiplikation von zwei Matrizen

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)

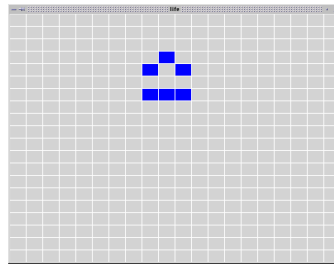


Abbildung 25.3: Population nach zwei Schritten

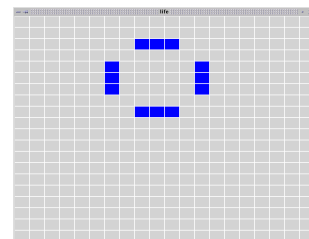
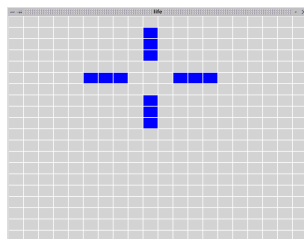


Abbildung 25.4: Population nach einigen weiteren Schritten, die sich nun ständig wiederholen

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
```

## 25.3 Zusammenhänge zwischen Arrays und Zeigern

### 25.3.1 Gaußsche Eliminationsverfahren *Mathematik(5;<150)*

Eine der fundamentalsten wissenschaftlichen Berechnungen ist die Lösung von Systemen von Gleichungen, die gleichzeitig erfüllt sein sollen. Der grundlegende Algorithmus für die Lösung derartiger Gleichungssysteme, das *Gaußsche Eliminationsverfahren*, ist relativ einfach und hat sich während der 150 Jahre seit seiner Entdeckung kaum verändert. Nehmen wir z.B. an, dass die folgenden drei Gleichungen vorliegen:

$$\begin{aligned}x_1 + 3x_2 - 4x_3 &= 8 \\x_1 + x_2 - 2x_3 &= 2 \\-x_1 - 2x_2 + 5x_3 &= -1\end{aligned}$$

Es sollen nun solche Werte für die 3 Variablen  $x_1$ ,  $x_2$  und  $x_3$  gefunden werden, dass für diese Werte alle 3 Gleichungen gleichzeitig erfüllt sind. In Abhängigkeit von den Gleichungen ist es möglich, dass dieses Problem keine Lösung hat (z.B. für  $x_1 + x_2 = 1$ ;  $x_1 + x_2 = 2$ ), oder dass viele Lösungen existieren (z.B. bei mehreren gleichen Gleichungen oder wenn mehr Variablen als Gleichungen vorhanden sind). Wir wollen hier voraussetzen, dass die Anzahl der Gleichungen und Variablen gleich ist, und einen Algorithmus betrachten, der eine eindeutige Lösung findet, wenn eine existiert. Um ein wiederholtes Schreiben der Variablen zu vermeiden, ist es zweckmäßig, eine Matrixschreibweise zu verwenden, um das Gleichungssystem auszudrücken. Die obigen Gleichungen lassen sich somit durch folgende Matrixgleichung angeben:

$$\begin{pmatrix} 1 & 3 & -4 \\ 1 & 1 & -2 \\ -1 & -2 & 5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 2 \\ -1 \end{pmatrix}$$

Es gibt verschiedene Operationen, die mit solchen Gleichungen ausgeführt werden können, ohne dass sie die Lösung ändern:

#### **Vertauschen von Gleichungen:**

Die Reihenfolge, in der die Gleichungen geschrieben werden, beeinflusst die Lösung nicht. Bei der Matrixdarstellung entspricht diese Operation einem Vertauschen von Zeilen der Matrix und den entsprechenden Elementen des Lösungsvektor auf der rechten Seite.

#### **Vertauschen von Variablen:**

Dies entspricht dem Vertauschen von Spalten in der Matrixdarstellung. Wenn Spalte  $i$  und  $j$  vertauscht werden, müssen auch die Variablen  $x_i$  und  $x_j$  vertauscht werden.

#### **Multiplizieren von Gleichungen mit einer Konstante:**

Dies entspricht in der Matrixdarstellung wiederum dem Multiplizieren einer Zeile der Matrix und des entsprechenden Elements des Lösungsvektors der rechten Seite mit einer Konstanten.

#### **Addieren bzw. Subtrahieren von zwei Gleichungen**

und Ersetzen einer dieser Gleichung durch die Summe.



Wir würden z.B. unter Anwendung der letztgenannten Operation auf unser obiges Gleichungssystem, indem wir die zweite Gleichung durch die Differenz der ersten beiden Gleichungen ersetzen, ein äquivalentes Gleichungssystem erhalten:

$$\begin{pmatrix} 1 & 3 & -4 \\ 0 & 2 & -2 \\ -1 & -2 & 5 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ -1 \end{pmatrix}$$

Beachten Sie, dass dadurch  $x_1$  aus der zweiten Gleichung eliminiert wird. Genauso können wir  $x_1$  aus der dritten Gleichung eliminieren, indem wir diese Gleichung durch die Summe der ersten und der dritten Gleichung ersetzen:

$$\begin{pmatrix} 1 & 3 & -4 \\ 0 & 2 & -2 \\ 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 8 \\ 6 \\ 7 \end{pmatrix}$$

Nun ist die Variable  $x_1$  aus allen Gleichungen außer der ersten eliminiert worden. Indem wir systematisch in dieser Weise vorgehen, können wir das ursprüngliche Gleichungssystem in ein System mit der gleichen Lösung umwandeln, das sich wesentlich leichter lösen lässt.

In unserem Beispiel ist dafür nur ein weiterer Schritt erforderlich, der zwei der obigen Operationen kombiniert: das Ersetzen der dritten Gleichung durch die Differenz der zweiten Gleichung und der mit 2 multiplizierten dritten Gleichung. So wird erreicht, dass alle Elemente unterhalb der Hauptdiagonale zu Nullen werden, und Gleichungssysteme dieser Form lassen sich besonders leicht lösen. Wenden wir diese beiden Operationen auf unser Gleichungssystem an, so erhalten wir die folgenden Gleichungen:

$$\begin{aligned} x_1 + 3x_2 - 4x_3 &= 8 \\ 2x_2 - 2x_3 &= 6 \\ -4x_3 &= -8 \end{aligned}$$

Die dritte Gleichung kann nun sofort gelöst werden:  $x_3 = 2$ . Wenn wir diesen Wert in der zweiten Gleichung einsetzen, können wir den Wert von  $x_2$  berechnen:

$$\begin{aligned} 2x_2 - 4 &= 6 \\ x_2 &= 5 \end{aligned}$$

In ähnlicher Weise gibt das Einsetzen dieser beiden Werte in die erste Gleichung die Möglichkeit, den Wert von  $x_1$  zu berechnen:

$$\begin{aligned} x_1 + 15 - 8 &= 8 \\ x_1 &= 1 \end{aligned}$$

womit wir die Lösung zu diesem Gleichungssystem haben. Dieses Beispiel zeigte die beiden wesentlichen Schritte des Gaußschen Eliminationsverfahrens:

### 1. Vorwärts-Elimination

Zuerst eliminiert man durch Addition geeigneter Vielfacher der ersten Gleichung zu jeder der anderen Gleichungen die erste Variable in allen Gleichungen mit Ausnahme der ersten. Dies bedeutet, dass in der ersten Spalte außer dem ersten Element nur noch Nullen vorkommen.

Dann eliminiert man durch Addition geeigneter Vielfacher der zweiten Gleichung zu jeder der Gleichungen von der dritten bis zur letzten die zweite Variable in allen Gleichungen mit Ausnahme der ersten zwei. Dies bedeutet, dass in zweiten Spalte außer den beiden ersten Elementen nur noch Nullen vorkommen. Dann eliminiert man die dritte Variable in allen Gleichungen mit Ausnahme der ersten drei usw. Am Ende der Vorwärts-Elimination befinden sich unter der Hauptdiagonalen nur noch Nullen.

## 2. Rückwärts-Substitution

Hier werden unter Benutzung der während des 1. Schrittes (Vorwärts-Elimination) erzeugten Dreiecksmatrix die Werte rückwärts berechnet.

Erstellen Sie ein Programm *gauselim.c*, das dieses Eliminationsverfahren realisiert, wie z.B.:

```

Wieviele Gleichungen hat das System: 3 (↩)
Element a11: 1 (↩)
Element a12: 3 (↩)
Element a13: -4 (↩)
Element b1: 8 (↩)

Element a21: 0 (↩)
Element a22: 2 (↩)
Element a23: -2 (↩)
Element b2: 6 (↩)

Element a31: -1 (↩)
Element a32: -2 (↩)
Element a33: 5 (↩)
Element b3: -1 (↩)
Soll schrittweise Elimination angezeigt werden (j/n): j (↩)

Gleichungssystem:
  1.00   3.00  -4.00 |   8.00
  0.00   2.00  -2.00 |   6.00
 -1.00  -2.00   5.00 |  -1.00

1. Eliminationsschritt:
  1.00   3.00  -4.00 |   8.00
  0.00   2.00  -2.00 |   6.00
  0.00   1.00   1.00 |   7.00

2. Eliminationsschritt:
  1.00   3.00  -4.00 |   8.00
  0.00   2.00  -2.00 |   6.00
  0.00   0.00   2.00 |   4.00

3. Eliminationsschritt:
  1.00   3.00  -4.00 |   8.00
  0.00   2.00  -2.00 |   6.00
  0.00   0.00   2.00 |   4.00

Lösung:
x1 =    1
x2 =    5
x3 =    2

```

### 25.3.2 Folgen von Nullen und Einsen *Mathematik(4;<50)*

Erstellen Sie ein Programm *nulleins.c*, das zunächst eine zufällige Folge von Nullen und Einsen erzeugt. Diese Folge soll dann in weiteren Schritten wie folgt komprimiert werden: Aus zwei gleiche aufeinanderfolgende Ziffern wird eine 0 und aus zwei unterschiedliche aufeinanderfolgende Ziffern wird eine 1. Diese Schritte werden solange wiederholt, bis nur noch eine Ziffer übrigbleibt. Mögliche Abläufe des Programms *nulleins.c* sind:

```
Länge der 0/1-Folge (max. 1000): 9 ↩
100000001
10001
101
11
0
```

```
Länge der 0/1-Folge (max. 1000): 71 ↩
01101011110011000101110110110100110001011100000001010100100101000100011
111000001101101000110000111011101011
010001110000010110
101000111
11001
001
01
1
```

### 25.3.3 Modus (Modalwert) einer Zahlenreihe *Statistik(4;<80)*

Wenn man eine Stichprobe hat, so wird oft der Wert gesucht, der am häufigsten in der Stichprobe vorkommt. Dieser Wert heißt auch *Modus* oder *Modalwert*. Erstellen Sie ein C-Programm *modus.c*, das zunächst ein Array mit zufälligen Zahlenwerten (von 0..99) füllt, bevor es dieses Array dann mittels `qsort()` sortiert und anschließend den Modus dieser Zahlenreihe ermittelt.

```
Wieviele Zufallswerte (1..10000): 100 ↩
Erzeugte Zufallszahlen -----
 75  78  40  36  79  14  7  83  96  75  11  39  53  78  69
57  45  7  34  19  52  98  41  93  98  96  27  94  96  86
 8  96  2  52  26  27  97  25  20  50  82  70  32  32  47
48  52  9  65  42  90  31  93  92  50  91  35  40  20  88
32  2  47  34  21  22  33  10  4  48  95  54  40  36  43
92  8  19  48  47  52  30  10  96  93  61  78  45  53  35
11  59  25  31  54  78  86  98  15  34
Sortierte Zufallszahlen -----
 2  2  4  7  7  8  8  9  10  10  11  11  14  15  19
19 20 20 21 22 25 25 26 27 27 30 31 31 32 32
32 33 34 34 34 35 35 36 36 39 40 40 40 41 42
43 45 45 47 47 47 48 48 48 50 50 52 52 52 52
53 53 54 54 57 59 61 65 69 70 75 75 78 78 78
78 79 82 83 86 86 88 90 91 92 92 93 93 93 94
95 96 96 96 96 96 97 98 98 98
....Modus: 96 (Haeufigkeit: 5).....
```

## 25.4 Strings und char-Zeiger

### 25.4.1 Streichen eines Zeichens aus String *Allgemein(4;<40)*

Erstellen Sie ein Programm *bucstrei.c*, das aus einem String, der einzugeben ist, alle Vertreter eines Zeichens, das ebenfalls einzugeben ist, streicht.

Möglicher Ablauf des Programms *ibucstrei.c*:

```
Geben Sie eine Zeichenkette (max. 1000 Zeichen) ein:
Das ist das Haus vom Nikolaus
Geben Sie das zu loeschende Zeichen ein: a
.....Die neue Zeichenkette ist dann:
Ds ist ds Hus vom Nikolus
```

### 25.4.2 Ersetzen von Wörtern in einem String *Allgemein(5;<80)*

Erstellen Sie ein Programm *wortrepl.c*, das in einem String, der einzugeben ist, alle Vertreter eines Wortes, das ebenfalls einzugeben ist, durch einen anderen String (auch einzugeben) ersetzt. Dabei sollen nur ganze Wörter ersetzt werden, d.h. dass bei Wörtern, in denen das zu ersetzende Wort nur ein Teilstring ist, keine Ersetzung stattfindet. Möglicher Ablauf des Programms *wortrepl.c* ist:

```
Gib einen String ein:
Das Schwein ist ein Lebewesen und kein Stein
Welches Wort soll ersetzt werden: ein
Durch welches Wort soll dieses ersetzt werden: ein nuetzliches
.....Neuer String:
Das Schwein ist ein nuetzliches Lebewesen und kein Stein
Noch eine Ersetzung (j/n): j
Welches Wort soll ersetzt werden: Das
Durch welches Wort soll dieses ersetzt werden: kein
.....Neuer String:
kein Schwein ist ein nuetzliches Lebewesen und kein Stein
Noch eine Ersetzung (j/n): j
Welches Wort soll ersetzt werden: kein
Durch welches Wort soll dieses ersetzt werden: viel
.....Neuer String:
viel Schwein ist ein nuetzliches Lebewesen und viel Stein
Noch eine Ersetzung (j/n): n
```

### 25.4.3 Palindrome in einem Text finden *Allgemein(4;<50)*

Erstellen Sie ein Programm *palindro.c*, das in einem Text, der Zeile für Zeile einzugeben ist, alle Palindrome findet und nach jeder Zeile ausgibt. Palindrome sind Wörter, die sich sowohl von hinten wie von vorne gleich lesen. Möglicher Ablauf des Programms *palindro.c* ist:

```
Anna lehnt am Reliefpfeiler und winkt Otto zu ↵
.....Anna
.....Reliefpfeiler
.....Otto
Andere Beispiele sind: SOS und ErikaFeuertNurUntreueFakire ↵
.....SOS
.....ErikaFeuertNurUntreueFakire
(Strg-D)
```

### 25.4.4 Sichere Zahleneingabe *Allgemein(5;<150)*

Erstellen Sie sich eigene Funktionen `read_long()`, `read_ulong()` und `read_double()` zum Einlesen von **long**-, **unsigned long**- und **double**-Zahlen. Diese Funktionen sollen unerlaubte Eingaben abfangen und nicht – wie beim Einlesen mit `scanf()` – zum Programmabsturz führen. Erstellen Sie ein Programm *zahlein.c*, das in seiner `main()`-Funktion diese drei Funktionen zum Testen aufruft.

Möglicher Ablauf des Programms *zahlein.c*:

```
Gib eine ganze Zahl ein: 983278378783278327689398 ↵
.....'983278378783278327689398' ist keine ganze Zahl (Zahl zu gross)
Gib eine ganze Zahl ein: 2.3 ↵
.....'2.3' ist keine ganze Zahl
Gib eine ganze Zahl ein: 793939Pa ↵
.....'793939Pa' ist keine ganze Zahl
Gib eine ganze Zahl ein: -9839838 ↵
= -9839838
Gib eine positive ganze Zahl ein: -89389378 ↵
.....'-89389378' ist keine positive ganze Zahl
Gib eine positive ganze Zahl ein: 8732737e3 ↵
.....'8732737e3' ist keine positive ganze Zahl
Gib eine positive ganze Zahl ein: 983289398 ↵
= 983289398
Gib eine Gleitpunktzahl ein: 3.4e-90933 ↵
.....'3.4e-90933' ist keine Gleitpunktzahl (zu klein)
Gib eine Gleitpunktzahl ein: 2.17 ↵
= 2.17
```

### 25.4.5 Wortlängen in einem Text *Allgemein(3;<40)*

Erstellen Sie ein Programm *wortlen.c*, das eine Häufigkeitsverteilung über die Wortlängen in einem Text ausgibt. Wendet man dieses Programm z.B. auf die Programmdatei *wortlen.c* selbst an: **wortlen <wortlen.c**

so könnte es z.B. die folgende Ausgabe liefern:

Wortlaenge	Anzahl
1	37
2	1
3	9
4	17
5	17
6	13
7	5
10	1
19	4

## 25.5 Array-Initialisierungen

### 25.5.1 Irren ist käferlich *Allgemein(4;<80)*

Die "Irrfahrt" eines Käfers auf den Kanten eines Würfels soll simuliert werden. Der Käfer startet in einer Ecke und wählt einen der drei möglichen Wege jeweils mit der gleichen Wahrscheinlichkeit. Erreicht er die diagonal gegenüberliegende Ecke, so ist die "Irrfahrt" zu Ende. Erstellen Sie ein Programm *kaefer.c*, das *n* solche "Irrfahrten" (*n* ist einzugeben) simuliert und dann ausgibt, welche Weglängen (Anzahl der durchlaufenen Kanten) mit welcher Häufigkeit durchlaufen wurden und wie groß die durchschnittliche Weglänge ist.

```

Wieviele Simulationen: 100000 (←)
|   Weglaenge   |   Wieoft   |
+-----+-----+
|           3   |   14716   |
|           5   |    9631   |
|           7   |    5540   |
|           9   |    4577   |
|          11   |    3503   |
|          13   |    2555   |
| .....        |
|          83   |         1   |
|          87   |         1   |
|
Durchschnittl. Weglaenge:   **** 8.64 Kanten ****

```

## 25.5.2 Wochentag zu bestimmten Datum *Allgemein(4;<50)*

Zur Lösung dieser Aufgabenstellung gibt es viele Methoden. Wir stellen hier eine in Form eines Pseudocodes vor:

```

jh_koeff[4] = { 4, 2, 0, 5 }
monat_koeff[12] = { 2, 5, 5, 1, 3, 6, 1, 4, 0, 2, 5, 0 }

Eingabe: tag, monat, jahr (wie z.B. 2.11.1997)
jh = Jahrhundert (vorderen beiden Ziffern der Jahreszahl)
ja = Jahr im Jahrhundert (hinteren beiden Ziffern der Jahreszahl)

if (monat > 2)
    schaltjahr = 0
else
    if (jahr%400==0)
        schaltjahr = 1
    else
        if (jahr%100==0)
            schaltjahr = 0
        else
            if (jahr%4==0)
                schaltjahr = 1
            else
                schaltjahr = 0
wochentag = (tag+monat_koeff[monat-1]-schaltjahr+jh_koeff[jh%4] + ja + ja/4 ) %7
Nr. in 'wochentag' ist Wochentag: (0=Sonntag, 1= Montag, ..., 6=Samstag)

```

Erstellen Sie zu diesem Pseudocode ein C-Programm *wochetag.c*. Mögliche Abläufe des Programms *wochetag.c* sind:

```
Gib Datum ein (tt.mm.jjjj): 12.2.1996 ↩
Das Datum 12.2.1996 ist ein Montag
```

```
Gib Datum ein (tt.mm.jjjj): 12.4.2045 ↩
Das Datum 12.4.2045 ist ein Mittwoch
```



### 25.5.3 Tagesnummer zu Datum und umgekehrt *Allgemein(4;<70)*

Erstellen Sie ein Programm *jahrtag.c*, das dem Benutzer, je nach Wahl, entweder die Tagesnummer zu einem Datum bzw. umgekehrt zu einem Datum die jeweilige Tagesnummer errechnet. Verwenden Sie in diesem Programm die folgende Initialisierung:

```
int monat_tage[][13] = {
    { 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 },
    { 0, 31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }
```

`monat_tage[0]` enthält dabei die Tage der Monate eines normalen Jahres und `monat_tage[1]` die in einem Schaltjahr.

Möglicher Ablauf des Programms *jahrtag.c*:

```
0  Ende
1  Tagesnummer zu einem Datum bestimmen
2  Datum zu einer Tagesnummer bestimmen

Deine Wahl: 1 (↔)

Gib dein Datum (tt.mm.jjjj) ein: 10.10.1998 (↔)
.... 10.10.1998 = 283. Tag im Jahr

0  Ende
1  Tagesnummer zu einem Datum bestimmen
2  Datum zu einer Tagesnummer bestimmen

Deine Wahl: 2 (↔)

Gib Tagesnummer und Jahr (nr,jahr) ein: 293,1998 (↔)
.... 293. Tag im Jahr = 20.10.1998

0  Ende
1  Tagesnummer zu einem Datum bestimmen
2  Datum zu einer Tagesnummer bestimmen

Deine Wahl: 0 (↔)
```

## 25.6 Zeigerarrays und Zeiger auf Zeiger

### 25.6.1 Interessante Wort-Zahlenfolgen *Allgemein(4;<50)*

Der Folge “einundzwanzig, dreizehn, acht, vier, vier” liegt ein interessantes Bildungsgesetz zugrunde. Welches? Versuchen Sie es herauszubekommen und erstellen Sie dann ein Programm *zahlfolg.c*, das dieses Bildungsgesetz auf einen Zahlenbereich anwendet, der vom Benutzer festzulegen ist, wie z.B:

```
Von wo: 2 
Bis wo: 22 

zwei, vier, vier
drei, vier, vier
vier, vier
fünf, vier, vier
sechs, fünf, vier, vier
sieben, sechs, fünf, vier, vier
acht, vier, vier
neun, vier, vier
zehn, vier, vier
elf, drei, vier, vier
zwölf, fünf, vier, vier
dreizehn, acht, vier, vier
vierzehn, acht, vier, vier
fünfzehn, acht, vier, vier
sechzehn, acht, vier, vier
siebzehn, acht, vier, vier
achtzehn, acht, vier, vier
neunzehn, acht, vier, vier
zwanzig, sieben, sechs, fünf, vier, vier
einundzwanzig, dreizehn, acht, vier, vier
zweiundzwanzig, vierzehn, acht, vier, vier
```

### 25.6.2 Das d'Hondtsche Höchstzählverfahren Allgemein(4;<80)

Gemäß Bundeswahlgesetz Paragraph 6 Abs. 1 werden die Sitze auf die Landeslisten nach dem dem „Höchstzählverfahren d'Hondt“ verteilt. Es heißt so nach seinem Urheber, dem Belgier Victor d'Hondt und wird in den Erläuterungen zum Bundeswahlgesetz wie folgt beschrieben:

*Dieses Höchstzählverfahren ist ein Rechenverfahren, durch das auf verhältnismäßig einfache Weise auf Grund der Stimmenzahl die proportionale Sitzverteilung ermittelt wird. Das Ergebnis entspricht nicht ganz, aber annähernd der mathematischen Proportion. Es besteht darin, dass die auf eine Wahlvorschlagsliste entfallenen Stimmen so oft durch 1, 2, 3 usw. geteilt werden, bis so viele 'Höchstzahlen' ermittelt sind, als Sitze zu verteilen sind. In der Reihenfolge der so ermittelten Höchstzahlen werden jeder Partei dann die Sitze zugewiesen.*

Angenommen, es seien insgesamt abgegeben:

für die Liste A	650500 Stimmen
B	541600 Stimmen
C	461500 Stimmen
D	89200 Stimmen
E	64800 Stimmen

und es seien insgesamt 31 Abgeordnete zu wählen. Folgende Tabelle enthält dann die Quotienten und in Klammern die Reihenfolge der Sitzzuteilung:

	A	B	C	D	E
	650500 (1)	541600 (2)	461500 (3)	89200 (19)	64800 (27)
durch 2	325250 (4)	270800 (5)	230750 (6)	44600	32400
durch 3	216833 (7)	180533 (8)	153833 (10)		
durch 4	162625 (9)	135400 (11)	115375 (13)		
durch 5	130100 (12)	108320 (15)	92300 (17)		
.....					

Würde man diese Rechnung fortsetzen, dann erhält die Partei A 12 Sitze, B erhält 9, C erhält 8 und D und E erhalten je einen Sitz.

Erstellen Sie nun ein C-Programm *dhondt.c*, das für Bundestags-, Landtags- und Kommunalwahlen die Sitzverteilung nach dem d'Hondtschen Höchstzählverfahren berechnet. Die Parteien seien dabei fest im Programm vorgegeben.

Möglicher Ablauf des Programms *dhondt.c* ist z.B.:

```
Geben Sie die abgegebenen Stimmen pro Partei ein
ZTU: 650500 (↵)
SPT: 541600 (↵)
FTB: 461500 (↵)
Greens: 89200 (↵)
Republiks: 64800 (↵)
Wieviele Sitze sind zu vergeben: 31 (↵)
Soll jede einzelne Sitzzuteilung ausgegeben werden (j/n) ? j (↵)
...Sitz 1 geht an: ZTU (650500)
...Sitz 2 geht an: SPT (541600)
...Sitz 3 geht an: FTB (461500)
...Sitz 4 geht an: ZTU (325250)
```

```

...Sitz 5 geht an: SPT (270800)
...Sitz 6 geht an: FTB (230750)
...Sitz 7 geht an: ZTU (216833)
...Sitz 8 geht an: SPT (180533)
...Sitz 9 geht an: ZTU (162625)
...Sitz 10 geht an: FTB (153833)
...Sitz 11 geht an: SPT (135400)
...Sitz 12 geht an: ZTU (130100)
...Sitz 13 geht an: FTB (115375)
...Sitz 14 geht an: ZTU (108417)
...Sitz 15 geht an: SPT (108320)
...Sitz 16 geht an: ZTU (92929)
...Sitz 17 geht an: FTB (92300)
...Sitz 18 geht an: SPT (90267)
...Sitz 19 geht an: Greens (89200)
...Sitz 20 geht an: ZTU (81312)
...Sitz 21 geht an: SPT (77371)
...Sitz 22 geht an: FTB (76917)
...Sitz 23 geht an: ZTU (72278)
...Sitz 24 geht an: SPT (67700)
...Sitz 25 geht an: FTB (65929)
...Sitz 26 geht an: ZTU (65050)
...Sitz 27 geht an: Reubliks (64800)
...Sitz 28 geht an: SPT (60178)
...Sitz 29 geht an: ZTU (59136)
...Sitz 30 geht an: FTB (57688)
...Sitz 31 geht an: ZTU (54208)

```

Sitzverteilung:

Partei	Sitze	Prozent (Sitze)	Prozent (Stimmen)
ZTU	12	38.71	35.99
SPT	9	29.03	29.96
FTB	8	25.81	25.53
Greens	1	3.23	4.93
Reubliks	1	3.23	3.58

### 25.6.3 Finden von Zahlwörtern in Strings *Allgemein(4;<50)*

*Endreim, Kurzweil, Nachtfalter, Wohnviertel, Neunauge, Weinstein, Erdreich, Achtung, Segelflieger, Pfalzwein, Radreifen, Gehhelfer, Leinsamen.* Alles zusammen macht 66.

Erstellen Sie ein Programm *wortadd.c*, das diese Aufgabe löst, wie z.B.:

Endreim: ...	drei ...	3
Kurzweil: ...	zwei ...	2
Nachtfalter: ...	acht ...	8
Wohnviertel: ...	vier ...	4
Neunauge: ...	neun ...	9
Weinstein: ...	eins ...	1
Erdreich: ...	drei ...	3
Achtung: ...	acht ...	8
Segelflieger: ...	elf ...	11
Pfalzwein: ...	zwei ...	2
Radreifen: ...	drei ...	3
Gehhelfer: ...	elf ...	11
Leinsamen: ...	eins ...	1
-----		
66		

### 25.6.4 Mischen von Farben *Allgemein(4;<50)*

Weißes Licht setzt sich aus verschiedenfarbigen Licht zusammen. Überlagert man die drei Grundfarben Rot, Grün und Violett, so entsteht weißes Licht. Mischt man zwei Grundfarben, so entsteht eine Mischfarbe.

Erstellen Sie ein Programm *farben.c*, das zunächst zwei Grundfarben einliest, und dann die entsprechende Mischfarbe gemäß der folgenden Tabelle ausgibt:

	Grün	Rot	Violett
Grün	Grün	Gelb	Blau
Rot	Gelb	Rot	Purpur
Violett	Blau	Purpur	Violett

Möglicher Ablauf des Programm *farben.c*:

```

Erste Grundfarbe (Grün Rot Violett ): Violett
..... Unbekannte Grundfarbe Violett (Neue Eingabe machen)
Erste Grundfarbe (Grün Rot Violett ): Violett
Zweite Grundfarbe (Grün Rot Violett ): Rot
..... Die Mischfarbe ist dann Purpur

```

# Kapitel 26

## Argumente auf der Kommandozeile

### 26.1 Dezimalzahlen nach Dual, Oktal und Hexa *Informatik(4;<80)*

Erstellen Sie ein Programm *zahlsys.c*, das die auf der Kommandozeile angegebene Dezimalzahl in folgende Darstellungen konvertiert:

```
--b Binaersystem
--o Oktalsystem
--h Hexadezimalsystem
```

Gruppieren der Optionen, wie z.B. `--hb` soll erlaubt sein. Sind keine Optionen angegeben, entspricht dies `--boh`. Die Optionen können vor oder nach der Zahl beliebig angegeben werden.

```
user@linux: > zahlsys --bo 127 --h
Dezimal: 127
Dual: 00000000 00000000 00000000 01111111
Oktal: 177
Hexadezimal: 7f
```

```
user@linux: > zahlsys --h --d 239 126
..... Unerlaubte Option --d
usage: zahlsys [--boh] zahl
..... Es muss genau eine Zahl angegeben sein
```

```
user@linux: > zahlsys --h --b 239
Dezimal: 239
Dual: 00000000 00000000 00000000 11101111
Hexadezimal: ef
```

```
user@linux: > zahlsys 237378383
Dezimal: 237378383
Dual: 00001110 00100110 00011011 01001111
Oktal: 1611415517
Hexadezimal: e261b4f
```

```
user@linux: > zahlsys --b -7837873 --h --hb
Dezimal: -7837873
Dual: 11111111 10001000 01100111 01001111
Hexadezimal: ff88674f
```

## 26.2 Rechnen mit Dualzahlen *Informatik(4;<60)*

Erstellen Sie ein Programm *dualrech.c*, das man mit einem dualen Ausdruck auf der Kommandozeile aufruft und das dann das entsprechende Ergebnis zu diesem Ausdruck ausgibt, wie z.B.

```
user@linux: > dualrech ↵
Richtiger Aufruf: dualrech <operand> <operator> <operand>
Erlaubte Operatoren sind: +, -, *, /, &, ^
```

```
user@linux: > dualrech 1000 - 10000 ↵
1000 - 10000 =
.....11111111111111111111111111111111000 (0xFFFFFFFF8)
```

```
user@linux: > dualrech 110011001o ; 10001120 ↵
.....110011001o ist keine erlaubte Dualzahl
.....10001120 ist keine erlaubte Dualzahl
.....; ist kein erlaubter Operator
```

```
user@linux: > dualrech 111011000000000111 ^ 11111111111100111111 ↵
111011000000000111 ^ 11111111111100111111 =
.....11000100111100111000 (0xC4F38)
```

# Kapitel 27

## Dynamische Speicher-Reservierung und -Freigabe

### 27.1 Numerierte oder rückwärtige Textausgabe *Informatik(5;<60)*

Erstellen Sie ein Programm *numrueck.c*, das Textzeilen einliest, diese im Speicher hält und dann bei Angabe der Option **-r** in umgekehrter Reihenfolge, sonst in der eingelesenen Reihenfolge wieder ausgibt. Ist die Option **-n** angegeben, so sind bei der Ausgabe den Zeilen die entsprechenden Nummern voranzustellen. Dieses Programm soll Speicherplatz nach Bedarf anfordern. Das bedeutet, dass auch das Zeiger-Array, das die Adressen der gespeicherten Zeilen enthält, nicht statisch anzulegen ist, sondern dass es dynamisch wachsen soll.

### 27.2 Strichliste *Informatik(4;<40)*

Erstellen Sie ein Programm *strilist.c*, das Ziffernfolgen (bis EOF) einliest und dann eine Strichliste über die Häufigkeit der einzelnen Ziffern ausgibt. Die einzelnen Strichlisten sind dabei als String zu speichern, der immer dynamisch zu erweitern ist. Zu Übungszwecken sollte dabei aber nicht `realloc()` verwendet werden, sondern der entsprechende Speicherplatz erst mit `free()` freigegeben werden, bevor mit `malloc()` seine neue Größe allokiert wird.

```
673276783298327876327632642876326237886276426 ⌅
736732636090009843984783263676532532432543263262187270 ⌅
Strg-D
0: |||||
1: |
2: |||||
.....
.....
8: |||||
9: ||||
```



## 27.3 Berechnen von Primzahlen *Mathematik(4;<60)*

Erstellen Sie ein Programm *primza.c*, das die Primzahlen zwischen 1 und  $n$  ( $n$  ist einzugeben) nach dem *Sieb des Eratosthenes* berechnet. Dieser Algorithmus ist sehr speicheraufwendig, da er zunächst alle natürlichen Zahlen zwischen 1 und  $n$  speichert, bevor er alle Nicht-Primzahlen aus dem Speicher streicht. Ihr Programm soll zunächst dynamisch Speicher für 100 Werte (Primzahlen bis 100) reservieren. Wenn dieser Speicherplatz nicht ausreicht, soll es mit `realloc()` den zuvor reservierten Speicherplatz immer wieder vergrößern.

Möglicher Ablauf des Programms *primza.c*:

Bis wohin sollen die Primzahlen berechnet werden (Ende=0) ? <b>70</b>									
2	3	5	7	11	13	17	19	23	29
31	37	41	43	47	53	59	61	67	
Bis wohin sollen die Primzahlen berechnet werden (Ende=0) ? <b>600</b>									
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	
Bis wohin sollen die Primzahlen berechnet werden (Ende=0) ? <b>10000</b>									
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
.....									
.....									
9539	9547	9551	9587	9601	9613	9619	9623	9629	9631
9643	9649	9661	9677	9679	9689	9697	9719	9721	9733
9739	9743	9749	9767	9769	9781	9787	9791	9803	9811
9817	9829	9833	9839	9851	9857	9859	9871	9883	9887
9901	9907	9923	9929	9931	9941	9949	9967	9973	
Bis wohin sollen die Primzahlen berechnet werden (Ende=0) ? <b>0</b>									

# Kapitel 28

## Strukturen

### 28.2 Operationen mit Strukturen

#### 28.2.1 Suchen eines optimalen Hauses *Allgemein(5;<200)*

Es ist ein Programm *hauskauf.c* zu erstellen, das zunächst die gewünschten Kriterien für ein Haus einliest und zwar die Unter- und Obergrenze zu folgenden Punkten: *Quadratmeter-Zahl*, *Grundstücksgröße*, *Zimmerzahl*, *Baujahr* und *Preis*. Bei jedem Punkt soll der Benutzer noch eingeben, ob er in diesem Intervall mehr zur Unter- oder zur Obergrenze tendiert. Nach der Eingabe der gewünschten Kriterien soll das Programm die Daten von angebotenen Häusern einlesen. Nach der Eingabe jedes Hauses soll sofort ausgegeben werden, ob dieses Haus die geforderten Kriterien erfüllt. Am Ende des Programms soll noch das optimalste unter Berücksichtigung der zu Beginn festgelegten Tendenzen ausgegeben werden, wie das folgende Ablaufbeispiel zeigt:

```
Minimale Quadratmeter-Zahl: 100 (←)
Maximale Quadratmeter-Zahl: 200 (←)
Bevorzugt (minimal=0; maximal=1): 0 (←)
Minimale Grundstuecksgroesse: 250 (←)
Maximale Grundstuecksgroesse: 1000 (←)
Bevorzugt (minimal=0; maximal=1): 1 (←)
Minimale Zimmerzahl: 4 (←)
Maximale Zimmerzahl: 10 (←)
Bevorzugt (minimal=0; maximal=1): 0 (←)
Minimales Baujahr: 1960 (←)
Maximales Baujahr: 1997 (←)
Bevorzugt (minimal=0; maximal=1): 1 (←)
Minimaler Preis: 200000 (←)
Maximaler Preis: 550000 (←)
Bevorzugt (minimal=0; maximal=1): 0 (←)
----Eingabe des 1.Hauses----
Qm-Zahl (Abbruch mit 0): 120 (←)
Grundstuecksgroesse: 450 (←)
Zimmerzahl: 5 (←)
```

```

Baujahr:          1985 (←)
Preis:            320000 (←)
.....Dieses Haus kommt in Frage

----Eingabe des 2.Hauses----
Qm-Zahl (Abbruch mit 0):  180 (←)
Grundstuecksgroesse:     650 (←)
Zimmerzahl:             7 (←)
Baujahr:              1979 (←)
Preis:                500000 (←)
.....Dieses Haus kommt in Frage

----Eingabe des 3.Hauses----
Qm-Zahl (Abbruch mit 0):  170 (←)
Grundstuecksgroesse:     559 (←)
Zimmerzahl:             8 (←)
Baujahr:              1981 (←)
Preis:                570000 (←)

----Eingabe des 4.Hauses----
Qm-Zahl (Abbruch mit 0):  110 (←)
Grundstuecksgroesse:     621 (←)
Zimmerzahl:             4 (←)
Baujahr:              1984 (←)
Preis:                340000 (←)
.....Dieses Haus kommt in Frage

----Eingabe des 5.Hauses----
Qm-Zahl (Abbruch mit 0):  110 (←)
Grundstuecksgroesse:     621 (←)
Zimmerzahl:             4 (←)
Baujahr:              1984 (←)
Preis:                340000 (←)
.....Dieses Haus kommt in Frage

----Eingabe des 7.Hauses----
Qm-Zahl (Abbruch mit 0):  130 (←)
Grundstuecksgroesse:     660 (←)
Zimmerzahl:             7 (←)
Baujahr:              1982 (←)
Preis:                210000 (←)
.....Dieses Haus kommt in Frage

----Eingabe des 8.Hauses----
Qm-Zahl (Abbruch mit 0):  0 (←)
.....Von 7 eingegeb. Haeusern ist folg. sehr gut geeignet....
Nr      :  4
qm      :  110 (100 - 200) <--
Grund   :  621 (250 - 1000) -->
Zimmer  :  4 (4 - 10) <--
Baujahr :  1984 (1960 - 1997) -->
Preis   :  340000 (200000 - 550000) <--
..... Auch sind die Haeuser mit folg. Nrn sehr gut geeignet....
5

```

### 28.2.2 Addieren von deutschen Kommazahlen *Allgemein(5;<80)*

Es soll ein Programm *kommaadd.c* erstellt werden, das das Addieren deutscher Kommazahlen ermöglicht. Der Vorkomma- und der Nachkommateil soll dabei in unterschiedlichen Komponenten einer Struktur gehalten und auch getrennt aufaddiert werden. Vor der Ausgabe ist eventuell ein entsprechender Übertrag aus dem Nachkommateil in den Vorkommateil notwendig.

```
Gib Deine Kommazahlen ein (Abschluss mit Leerzeile)
3,4 ↩
3,4456 ↩
123,23 ↩
2,3333 ↩
77,8 ↩
34 ↩
7,0 ↩
2 ↩
3,123456 ↩
1,5 ↩
1,23 ↩
↩
= 259,062356 (256, 3062356) .....Kontrollwert: 259.062356
```



### 28.4.3 Umwandeln arabischer in römische Zahlen *Allgemein(4;<50)*

Erstellen Sie ein Programm *romzahl.c*, das eine arabische Zahl einliest und dann die zugehörige römische Zahl ausgibt. Hierzu einige Tipps:


Folgende römischen Ziffernkombinationen stehen für die entsprechenden arabischen Zahlen:


"I",	1,
"IV",	4,
"V",	5,
"IX",	9,
"X",	10,
"XL",	40,
"L",	50,
"XC",	90,
"C",	100,
"CD",	400,
"D",	500,
"CM",	900,
"M",	1000

Diese Zuordnung lässt sich leicht mit einem Strukturarray lösen, das man schon bei der Definition mit obigen Werten initialisiert. Hat man nun eine arabische Zahl, dann lässt sich die zugehörige römische Zahl leicht mit folgendem Algorithmus ermitteln:


```
for (i=Index des letzten Elements des Strukturarrays; i>=0; i--) {
    for (j=0; j<zahl/(i-ten arabischen Wert); j++)
        Ausgabe der zugehörigen i-ten römischen Zifferkombination;
    zahl %= (i-ten arabischen Wert);
}
```


Mögliche Abläufe des Programms *romzahl.c*:

Zu wandelnde Zahl: **1994**   
.... 1994 = MCMXCIV ....

Zu wandelnde Zahl: **999**   
.... 999 = CMXCIX ....

Erstellen Sie zusätzlich noch ein Programm *romzahl2.c* (lediglich Erweiterung des Strukturarrays gegenüber *romzahl.c*), das die kürzest mögliche römische Zahl zu einer arabischen Zahl ausgibt, wie z.B.:

Zu wandelnde Zahl: **1994**   
.... 1994 = MXMIV ....

Zu wandelnde Zahl: **999**   
.... 999 = IM ....

### 28.4.4 Notenübersicht und Notenspiegel *Allgemein(3;<60)*

Ein Lehrer braucht ein Programm *noteverw.c*, das ihm die Notenübersicht für Klassenarbeiten erleichtert. Dieses Programm soll zunächst für alle Schüler einer Klasse den Namen, Vornamen und die in der Klassenarbeit erreichte Note einlesen, bevor es dann eine Namensliste mit jeweils erreichter Note, die Durchschnittsnote und einen Notenspiegel in Form eines Histogramms ausgibt, wie z.B.:

```

----Eingabe des 1.Schuelers----
Name:   Meier (←)
Vorname: Hans (←)
Note:   4 (←)
----Eingabe des 2.Schuelers----
Name:   Mueller (←)
Vorname: Fritz (←)
Note:   3 (←)
----Eingabe des 3.Schuelers----
Name:   Haller (←)
Vorname: Antonia (←)
Note:   1 (←)
----Eingabe des 4.Schuelers----
Name:   Feller (←)
Vorname: Marion (←)
Note:   3 (←)
----Eingabe des 5.Schuelers----
Name:   Goller (←)
Vorname: Axel (←)
Note:   2 (←)
----Eingabe des 6.Schuelers----
Name:   Kanter (←)
Vorname: Toni (←)
Note:   4 (←)
----Eingabe des 7.Schuelers----
Name:   Maller (←)
Vorname: Egon (←)
Note:   3 (←)
----Eingabe des 8.Schuelers----
Name:   Oller (←)
Vorname: Udo (←)
Note:   3 (←)
----Eingabe des 9.Schuelers----
Name:   Zanter (←)
Vorname: Karl (←)
Note:   4 (←)
.....
.....
.....
.....

```

----Eingabe des 23.Schuelers----

Name:



Name	, Vorname	, Note
------	-----------	--------

1. Meier	, Hans	, 4
2. Mueller	, Fritz	, 3
3. Haller	, Antonia	, 1
4. Feller	, Marion	, 3
5. Goller	, Axel	, 2
6. Kanter	, Toni	, 4
7. Maller	, Egon	, 3
8. Oller	, Udo	, 3
9. Zanter	, Karl	, 4
10. Reuter	, Wolfgang	, 3
11. Ammer	, Walter	, 2
12. Polter	, Michaela	, 3
13. Wolters	, Danni	, 5
14. Milber	, Dieter	, 6
15. Winter	, Tom	, 4
16. Baller	, Silke	, 3
17. Schwegler	, Gunter	, 5
18. Doller	, Hansi	, 2
19. Nibler	, Ingrid	, 3
20. Ranneberg	, Tanja	, 4
21. Elters	, Sascha	, 3
22. Vogts	, Mandy	, 4

....Durchschnittsnote: 3.29

....Notenspiegel

Note 1: \*

Note 2: \*\*\*

Note 3: \*\*\*\*\*

Note 4: \*\*\*\*\*

Note 5: \*\*

Note 6: \*



## 28.5 Strukturen als Funktionsparameter

### 28.5.1 Tagesdifferenz zwischen zwei Dat *Allgemein(5;<60)*

Die weltberühmte Hofbräuhütte wurde z.B. am 7.11.1902 gegründet und hatte seitdem jeden Tag geöffnet. Zum Hundertjährigen Jubiläum soll nun ein großes Fest veranstaltet werden, an dem die Hofbräuhütte für jeden Öffnungstag eine Maß Freibier ausschenkt. Hierzu muss dieses traditionelle und weltbekannte Gasthaus aber die Anzahl der Tage zwischen dem 7.11.1902 und dem 7.11.2002 ermitteln.

Helfen Sie der Hofbräuhütte und erstellen Sie ein Programm *datediff.c*, das zwei Dat einliest und dann die Anzahl der Tage zwischen diesen beiden Dat ausgibt. Die eingegebenen Dat werden dabei in zwei Strukturvariablen vom gleichen Datentyp gespeichert. Zum Ermitteln der Tagesdifferenz sollten Sie eine Funktion `tag_diff()` erstellen, der sie als Argumente diese beiden Strukturvariablen übergeben.

```
1. Datum (tt.mm.jjjj): 7.11.1902 (←)
2. Datum (tt.mm.jjjj): 7.11.2002 (←)
..... Differenz: 36525 Tage .....
```

```
1. Datum (tt.mm.jjjj): 7.2.1996 (←)
2. Datum (tt.mm.jjjj): 7.3.1996 (←)
..... Differenz: 29 Tage .....
```

```
1. Datum (tt.mm.jjjj): 7.2.1700 (←)
2. Datum (tt.mm.jjjj): 7.3.1700 (←)
..... Differenz: 28 Tage .....
```

### 28.5.2 Addition von zwei Zeiten *Allgemein(3;<60)*

Erstellen Sie ein Programm *zeitadd.c*, das zwei Zeiten addiert. In diesem Programm sollten Sie zwei Funktionen mit folgender Deklaration erstellen und auch verwenden:

```
unsigned long  zeit_in_sek(struct zeit z);
struct zeit    sek_in_zeit(unsigned long sek);
```

Die Struktur `zeit` könnte z.B. folgendes Aussehen haben:

```
struct zeit    int tag;        int std;        int min;        int sek;        ;
```

Mögliche Abläufe des Programms *zeitadd.c*:

```
Gib 1. Zeit ein: (tt.hh.mm.ss): 1.5.23.49 (←)
Gib 2. Zeit ein: (tt.hh.mm.ss): 3.19.44.26 (←)

..... = 5 Tage, 1:8.15; 436095 Gesamtsekunden
```

```
Gib 1. Zeit ein: (tt.hh.mm.ss): 0.19.59.1 (←)
Gib 2. Zeit ein: (tt.hh.mm.ss): 0.4.1.5 (←)

..... = 1 Tage, 0:0.6; 86406 Gesamtsekunden
```

## 28.6 Zeiger und Strukturen

### 28.6.1 Mischen von zwei sortierten Listen *Informatik(4;<150)*

Erstellen Sie ein Programm *listmisc.c*, das Namen einliest und diese in eine von zwei unterschiedlichen Listen einsortiert. In welche Liste der jeweilige Namen einzusortieren ist, kann der Benutzer festlegen. Nach dem Eingabeende (Leerzeile) soll dieses Programm die zwei sortierten Listen zu einer gemischten sortierten Liste zusammenfassen, bevor es dann alle drei Listen ausgibt, wie z.B.

```
Name: Hans (↩)
    Welche Liste (1 oder 2) : 1 (↩)
Name: Emil (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Antonia (↩)
    Welche Liste (1 oder 2) : 1 (↩)
Name: Toni (↩)
    Welche Liste (1 oder 2) : 1 (↩)
Name: Fritz (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Emil (↩)
    Welche Liste (1 oder 2) : 1 (↩)
Name: Helmut (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Zorro (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Ulrike (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Wolfgang (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Albert (↩)
    Welche Liste (1 oder 2) : 1 (↩)
Name: Manfred (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Zander (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: Veronika (↩)
    Welche Liste (1 oder 2) : 2 (↩)
Name: (↩)
(↩)
....1. Liste.....
Albert
Antonia
Emil
Hans
Toni
....Listenende....
```

....2. Liste.....

Emil

Fritz

Helmut

Manfred

Ulrike

Veronika

Wolfgang

Zander

Zorro

....Listenende.....

....Gemischte. Liste.....

Albert

Antonia

Emil

Emil

Fritz

Hans

Helmut

Manfred

Toni

Ulrike

Veronika

Wolfgang

Zander

Zorro

....Listenende.....

## 28.6.2 Eine Liste, aber unterschiedlich sortiert *Informatik(4;,<120)*

Erstellen Sie ein Programm *doppsort.c*, das Namen und Telefonnummern einliest und diese in einer verketteten Liste einhängt. Diese Liste soll doppelt verkettet sein: Die Verkettung der Erstzeiger liefert die Namen in alphabetischer Ordnung und die Verkettung der Zweitzeiger die numerische Ordnung der Telefonnummern.

Möglicher Ablauf des Programms *doppsort.c*:

```
Namens- und Telefonliste
=====
N   Einfuegen neuer Namen mit Telefonnummer
A   Ausgeben der Liste
E   Ende                               Deine Wahl: n (←)

Name: Hans (←)
Tel: 12345 (←)
Name: Emil (←)
Tel: 23456 (←)
Name: Toni (←)
Tel: 11234 (←)
Name: (←)

N   Einfuegen neuer Namen mit Telefonnummer
A   Ausgeben der Liste
E   Ende                               Deine Wahl: a (←)

....Liste nach Namen sortiert.....
Emil (Tel. 23456)
Hans (Tel. 12345)
Toni (Tel. 11234)

....Liste nach Telefonnummern sortiert.....
11234 (Toni)
12345 (Hans)
23456 (Emil)

N   Einfuegen neuer Namen mit Telefonnummer
A   Ausgeben der Liste
E   Ende                               Deine Wahl: n (←)

Name: Fritz (←)
Tel: 66634 (←)
Name: Fritz (←)
Tel: 11122 (←)
Name: Helmut (←)
Tel: 99734 (←)
Name: Wolfgang (←)
Tel: 54321 (←)
```

```
Name: 

N  Einfuegen neuer Namen mit Telefonnummer
A  Ausgeben der Liste
E  Ende                               Deine Wahl: a 

....Liste nach Namen sortiert.....
Emil (Tel. 23456)
Fritz (Tel. 11122)
Fritz (Tel. 66634)
Hans (Tel. 12345)
Helmut (Tel. 99734)
Toni (Tel. 11234)
Wolfgang (Tel. 54321)

....Liste nach Telefonnummern sortiert.....
11122 (Fritz)
11234 (Toni)
12345 (Hans)
23456 (Emil)
54321 (Wolfgang)
66634 (Fritz)
99734 (Helmut)

N  Einfuegen neuer Namen mit Telefonnummer
A  Ausgeben der Liste
E  Ende                               Deine Wahl: e 
```

### 28.6.3 Wortstatistik zu einem Text *Allgemein(4;<80)*

Erstellen Sie ein Programm *wortstat.c*, das zu einem Text eine alphabetisch geordnete Wortstatistik erstellt, was bedeutet, dass zu jedem Wort die Häufigkeit seines Vorkommens im vorgelegten Text ausgegeben wird. Verwenden Sie zur Lösung dieser Aufgabenstellung einen Binärbaum. Hat man z.B. folgenden Text in der Datei *schwafel.txt*:

```
Heute sind die Voraussetzungen für eine gute Tat
sehr günstig; solche Voraussetzungen wünscht man sich.
Nur weiter so bis Ende des Jahres;
vielleicht auch einen guten Umsatz, der dreistellig ist;
dann werden sich auch schwarze Zahlen sehen lassen;
Jeder kann etwas dazu beitragen.

Heute wünscht man sich Voraussetzungen, die sich bis Ende
des Jahres günstig auswirken. So soll es sein und
wir wünschen einen guten Jahresabschluss.
Der Umsatz, der besser sein könnte, möge sich weiter erhöhen,
und dann sehen wir in eine rosige Zukunft.
Nun wollen wir das Jammern lassen und jeder soll das tun, was
er kann.
```

und man ruft das Programm *wortstat* mit Eingabeumlenkung auf, wie

```
wortstat < schwafel.txt
```

so sollte folgendes ausgegeben werden:

```
auch          :    2
auswirken     :    1
beitragen     :    1
besser        :    1
bis           :    2
dann          :    2
das           :    2
dazu          :    1
der           :    3
des           :    2
.....
.....
werden        :    1
wir           :    3
wollen        :    1
wünschen      :    1
wünscht       :    2
zahlen        :    1
zukunft       :    1
```

### 28.6.4 Realisierung eines Binärbaums mit Array *Informatik(4;<100)*

Erstellen Sie ein Programm *binarray.c*, das einen Binärbaum über ein Array realisiert. Statt mit Zeigern sollten Sie dabei mit Indizes für die linken und rechten Nachfolger arbeiten, wie z.B.

```
struct element {
    char  name[20];
    int   links;
    int   rechts;
} **liste=NULL;
```

Das Strukturarray *liste* soll dabei dynamisch bei jedem neuen Namen um ein Strukturelement verlängert werden. Zu Testzwecken sollte das Array einmal mit seinen Indizes und einmal mit den linken und rechten Nachkommens-Namen, auf die die Indizes verweisen, ausgegeben werden, bevor dann die alphabetische Reihenfolge unter Verwendung von Rekursion ausgegeben wird, wie z.B.

```
1. Name: Hans (←)
2. Name: Fritz (←)
3. Name: Christa (←)
4. Name: Emil (←)
5. Name: Albert (←)
6. Name: Toni (←)
7. Name: Wolfgang (←)
8. Name: Helmut (←)
9. Name: Zorro (←)
10. Name: Anton (←)
11. Name: Berta (←)
12. Name: (←)
```

Array als Binaerbaum mit entspr. Indizes fuer linke/rechte Nachfolger

```
=====
0.      Hans | 1 | 5 |
1.      Fritz | 2 | -1 |
2.      Christa | 4 | 3 |
3.      Emil | -1 | -1 |
4.      Albert | -1 | 9 |
5.      Toni | 7 | 6 |
6.      Wolfgang | -1 | 8 |
7.      Helmut | -1 | -1 |
8.      Zorro | -1 | -1 |
9.      Anton | -1 | 10 |
10.     Berta | -1 | -1 |
```

Array als Binaerbaum mit entspr. linke/rechte Nachfolger

=====

```

0.    Fritz <--- Hans    ---> Toni
1.    Christa <--- Fritz  ---> |
2.    Albert <--- Christa ---> Emil
3.      | <--- Emil    ---> |
4.      | <--- Albert  ---> Anton
5.    Helmut <--- Toni    ---> Wolfgang
6.      | <--- Wolfgang ---> Zorro
7.      | <--- Helmut  ---> |
8.      | <--- Zorro   ---> |
9.      | <--- Anton   ---> Berta
10.     | <--- Berta   ---> |

```

Alphabetische Ausgabe des Binaerbaum-Arrays

=====

```

1. Albert
2. Anton
3. Berta
4. Christa
5. Emil
6. Fritz
7. Hans
8. Helmut
9. Toni
10. Wolfgang
11. Zorro

```



## 28.7 Spezielle Strukturen (Unions und Bitfelder)

### 28.7.1 Partnersuche in einem Heiratsinstitut *Allgemein(4;<200)*

Ein Heiratsinstitut führt in einer Kartei paarungsrelevante Daten seiner Kunden. Erstellen Sie ein Programm *heirat.c*, das es erlaubt, neue Daten in die Kartei einzutragen, alle Daten aus der Kartei anzuzeigen oder zu einem bestimmten Kunden potentielle Partner zu suchen. Zwei Kunden kommen dann als Partner in Frage, wenn

- ☐ das Geschlecht verschieden und
- ☐ der Altersunterschied nicht mehr als 10 Jahre und
- ☐ die Schnittmenge der Interessen nicht leer sind.

Mögliche Interessen sind:

*Garten, Basteln, Sport, Literatur, Musik, Kunst, Reisen und Wandern*

Arbeiten sie im Programm *heirat.c* mit Bitfeldern.

Möglicher Ablauf des Programms *heirat.c*:

```

Heiratsinstitut
=====
0   Ende
1   Neuen Kunden eingeben
2   Zusammenpassende Kunden suchen
3   Alle Kunden anzeigen

Ihre Wahl: 1 (←)
.....Eingabe des 1. Kunden.....
Name      : Haller (←)
Vorname   : Fritz (←)
Postleitz.: 91091 (←)
Wohnort    : Kleinstadeln (←)
Strasse Nr: Bollerstr. 17 (←)
Alter     : 37 (←)
Geschlecht (maennlich=0, weiblich=1): 0 (←)
Hobbies
  Garten (ja=1, nein=0): 1 (←)
  Basteln (ja=1, nein=0): 0 (←)
  Sport (ja=1, nein=0): 1 (←)
  Literatur (ja=1, nein=0): 0 (←)
  Musik (ja=1, nein=0): 1 (←)
  Kunst (ja=1, nein=0): 0 (←)
  Reisen (ja=1, nein=0): 1 (←)
  Wandern (ja=1, nein=0): 0 (←)
.....Eingabe des 2. Kunden.....
Name      : Mueller (←)
Vorname   : Antonia (←)
Postleitz.: 12345 (←)
Wohnort    : Nordhausen (←)
Strasse Nr: Zeppelinstr. 12 (←)

```

```

Alter      : 29 (↵)
Geschlecht (maennlich=0, weiblich=1): 1 (↵)
Hobbies
  Garten (ja=1, nein=0): 1 (↵)
  Basteln (ja=1, nein=0): 0 (↵)
  Sport (ja=1, nein=0): 1 (↵)
  Literatur (ja=1, nein=0): 0 (↵)
  Musik (ja=1, nein=0): 1 (↵)
  Kunst (ja=1, nein=0): 0 (↵)
  Reisen (ja=1, nein=0): 1 (↵)
  Wandern (ja=1, nein=0): 0 (↵)
.....Eingabe des 3. Kunden.....
Name       : Faller (↵)
Vorname    : Fritz (↵)
Postleitz.: 53533 (↵)
Wohnort    : Glotzenbruck (↵)
Strasse Nr: Talstr. 55 (↵)
Alter      : 50 (↵)
Geschlecht (maennlich=0, weiblich=1): 0 (↵)
Hobbies
  Garten (ja=1, nein=0): 0 (↵)
  Basteln (ja=1, nein=0): 1 (↵)
  Sport (ja=1, nein=0): 0 (↵)
  Literatur (ja=1, nein=0): 1 (↵)
  Musik (ja=1, nein=0): 0 (↵)
  Kunst (ja=1, nein=0): 1 (↵)
  Reisen (ja=1, nein=0): 0 (↵)
  Wandern (ja=1, nein=0): 1 (↵)
.....Eingabe des 4. Kunden.....
Name       : Galler (↵)
Vorname    : Marion (↵)
Postleitz.: 74643 (↵)
Wohnort    : Tulpenstadt (↵)
Strasse Nr: Am Giesenberg 4 (↵)
Alter      : 32 (↵)
Geschlecht (maennlich=0, weiblich=1): 1 (↵)
Hobbies
  Garten (ja=1, nein=0): 0 (↵)
  Basteln (ja=1, nein=0): 0 (↵)
  Sport (ja=1, nein=0): 0 (↵)
  Literatur (ja=1, nein=0): 1 (↵)
  Musik (ja=1, nein=0): 1 (↵)
  Kunst (ja=1, nein=0): 0 (↵)
  Reisen (ja=1, nein=0): 0 (↵)
  Wandern (ja=1, nein=0): 0 (↵)

```

```

.....Eingabe des 5. Kunden.....
Name      : Prossel
Vorname   : Marco
Postleitz.: 63677
Wohnort   : Ostheim
Strasse Nr: Am Bach 7a
Alter     : 35
Geschlecht (maennlich=0, weiblich=1): 0
Hobbies
  Garten (ja=1, nein=0): 0
  Basteln (ja=1, nein=0): 0
  Sport (ja=1, nein=0): 0
  Literatur (ja=1, nein=0): 0
  Musik (ja=1, nein=0): 0
  Kunst (ja=1, nein=0): 0
  Reisen (ja=1, nein=0): 1
  Wandern (ja=1, nein=0): 0
.....Eingabe des 6. Kunden.....
Name      :
0      Ende
1      Neuen Kunden eingeben
2      Zusammenpassende Kunden suchen
3      Alle Kunden anzeigen

Ihre Wahl: 3
1. Haller, Fritz, 91091 Kleinstadeln, Bollerstr. 17
   maennlich, 37 Jahre
   Wandern, Kunst, Literatur, Basteln,
2. Mueller, Antonia, 12345 Nordhausen, Zeppelinstr. 12
   weiblich, 29 Jahre
   Wandern, Kunst, Literatur, Basteln,
3. Faller, Fritz, 53533 Glotzenbruck, Talstr. 55
   maennlich, 50 Jahre
   Reisen, Musik, Sport, Garten,
4. Galler, Marion, 74643 Tulpenstadt, Am Giesenberg 4
   weiblich, 32 Jahre
   Musik, Literatur,
5. Prossel, Marco, 63677 Ostheim, Am Bach 7a
   maennlich, 35 Jahre
   Basteln,

0      Ende
1      Neuen Kunden eingeben
2      Zusammenpassende Kunden suchen
3      Alle Kunden anzeigen

Ihre Wahl: 2

```

```

1. Haller, Fritz, 91091 Kleinstadeln, Bollerstr. 17
   maennlich, 37 Jahre
   Wandern, Kunst, Literatur, Basteln,
2. Mueller, Antonia, 12345 Nordhausen, Zeppelinstr. 12
   weiblich, 29 Jahre
   Wandern, Kunst, Literatur, Basteln,
3. Faller, Fritz, 53533 Glotzenbruck, Talstr. 55
   maennlich, 50 Jahre
   Reisen, Musik, Sport, Garten,
4. Galler, Marion, 74643 Tulpenstadt, Am Giesenberg 4
   weiblich, 32 Jahre
   Musik, Literatur,
5. Prossel, Marco, 63677 Ostheim, Am Bach 7a
   maennlich, 35 Jahre
   Basteln,

```

.....Partner zu welchen Kunden suchen (1..5): 2 

Zu dem Kunden

```

-----
2. Mueller, Antonia, 12345 Nordhausen, Zeppelinstr. 12
   weiblich, 29 Jahre
   Wandern, Kunst, Literatur, Basteln,
-----

```


passen die folgenden Kunden:

```

1. Haller, Fritz, 91091 Kleinstadeln, Bollerstr. 17
   maennlich, 37 Jahre
   Wandern, Kunst, Literatur, Basteln,
5. Prossel, Marco, 63677 Ostheim, Am Bach 7a
   maennlich, 35 Jahre
   Basteln,
-----

```

- 0 Ende
- 1 Neuen Kunden eingeben
- 2 Zusammenpassende Kunden suchen
- 3 Alle Kunden anzeigen

Ihre Wahl: 2 

```

1. Haller, Fritz, 91091 Kleinstadeln, Bollerstr. 17
   maennlich, 37 Jahre
   Wandern, Kunst, Literatur, Basteln,
2. Mueller, Antonia, 12345 Nordhausen, Zeppelinstr. 12
   weiblich, 29 Jahre
   Wandern, Kunst, Literatur, Basteln,
3. Faller, Fritz, 53533 Glotzenbruck, Talstr. 55
   maennlich, 50 Jahre
   Reisen, Musik, Sport, Garten,
4. Galler, Marion, 74643 Tulpenstadt, Am Giesenberg 4
   weiblich, 32 Jahre

```

```
Musik, Literatur,
5. Prossel, Marco, 63677 Ostheim, Am Bach 7a
maennlich, 35 Jahre
Basteln,
.....Partner zu welchen Kunden suchen (1..5): 4 (←)
Zu dem Kunden
-----
4. Galler, Marion, 74643 Tulpenstadt, Am Giesenberg 4
weiblich, 32 Jahre
Musik, Literatur,
-----
passen die folgenden Kunden:
1. Haller, Fritz, 91091 Kleinstadeln, Bollerstr. 17
maennlich, 37 Jahre
Wandern, Kunst, Literatur, Basteln,
-----

0   Ende
1   Neuen Kunden eingeben
2   Zusammenpassende Kunden suchen
3   Alle Kunden anzeigen
Ihre Wahl: 0 (←)
```

# Kapitel 29

## Eigene Datentypen

### 29.1 Mischtablette aus der Chemie *Chemie(4;<60)*

Erstellen Sie ein Programm *laugsaeu.c*, das folgende Mischtablette ausgibt:

---

	Salzsäure	Schwefelsäure	Salpetersäure	Kohlensäure
-----	-----	-----	-----	-----
Natronlauge	Natriumchlorid	Natriumsulfat	Natriumnitrat	Natriumcarbonat
Kalilauge	Kaliumchlorid	Kaliumsulfat	Kaliumnitrat	Kaliumcarbonat
Kalkwasser	Calciumchlorid	Calciumsulfat	Calciumnitrat	Calciumcarbonat

---

Verwenden Sie im Programm *laugsaeu.c* zu Übungszwecken die Schlüsselwörter **enum** und **typedef**!



# Kapitel 30

## Dateien

### 30.1 Höhere E/A-Funktionen

#### 30.1.1 Ausgeben der letzten $n$ Zeilen einer Datei *Allgemein(4;<60)*

Erstellen Sie ein Programm *lastline.c*, das die letzten  $n$  Zeilen einer Textdatei, deren Name auf der Kommandozeile anzugeben ist, ausgibt. Der mögliche Aufruf von *lastline* ist:

**lastline -n *dateiname***

Die Angabe von **-n** ( $n$  ist eine Zahl) ist dabei optional. Ist **-n** nicht angegeben, so sollen die letzten 10 Zeilen der Datei *dateiname* ausgegeben werden.

Bei der Realisierung des Programms *lastline.c* empfiehlt es sich, die Textdatei zuerst vollständig (zeilenweise) zu lesen, und sich die Offsets der einzelnen Zeilen in einem **int**-Array zu merken. Bei der Ausgabe der letzten Zeilen sollte dann unter Verwendung von `fseek()` auf die entsprechenden Zeilen positioniert werden.

```
user@linux: > lastline lastline.c ↵
43     fseek(fz, offset[i], SEEK_SET);
44     if (fgets(zeile, MAX_LAENGE, fz) != NULL)
45         fprintf(stdout, "%5d  %s", i, zeile);
46     else {
47         fprintf(stderr, "....Lesefehler in Datei '%s'\n", dateiname);
48         exit(4);
49     }
50 }
51 return(0);
52 }

user@linux: > lastline -5 lastline.c ↵
48     exit(4);
49 }
50 }
51 return(0);
52 }
```



### 30.1.2 Ausgeben einer Datei mit Zeilennummerierung *Allgemein(3;<30)*

Erstellen Sie ein Programm *numausg.c*, das eine Textdatei, deren Name als erstes Argument auf der Kommandozeile anzugeben ist, mit Zeilennummerierung am Bildschirm ausgibt.

### 30.1.3 Erzeugen von Adress-Aufkleber *Allgemein(3;<50)*

Erstellen Sie ein C-Programm *adrdruck.c*, das eine Adresse einliest und diese Adresse dann *n* mal in die Datei *adresse.txt* schreibt (*n* ist einzugeben), wie z.B.:

```
Dieses Programm liest eine Adresse ein und schreibt
diese Adresse n mal in die Datei 'adresse.txt'

Vorname      : Alexander
Nachname     : Fleischmann
Strasse      : Bergstrasse
Hausnummer   : 172a
Postleitzahl : 50636
Wohnort      : Nordenhofen
Telefon      : 0738/88327737
Fax          : 0738/78827740
Wie oft soll Adresse in Datei geschrieben werden: 4
```

Nach Ablauf dieses Programms hat die Datei *adresse.txt* den folgenden Inhalt:

```
Alexander Fleischmann
Bergstrasse 172a
50636 Nordenhofen

Tel. 0738/88327737
Fax 0738/78827740
-----
Alexander Fleischmann
Bergstrasse 172a
50636 Nordenhofen

Tel. 0738/88327737
Fax 0738/78827740
-----
Alexander Fleischmann
Bergstrasse 172a
50636 Nordenhofen

Tel. 0738/88327737
Fax 0738/78827740
-----
Alexander Fleischmann
Bergstrasse 172a
50636 Nordenhofen

Tel. 0738/88327737
Fax 0738/78827740
```

### 30.1.4 Suchen von Strings in Dateien *Allgemein(3;<30)*

Erstellen Sie ein C-Programm *suchtext.c*, das nach einem auf der Kommandozeile angegebenen Text in einer Datei sucht. Der zu suchende Text ist dabei als erstes Argument und die Datei, in der zu suchen ist, als zweites Argument anzugeben. Nachfolgend sind einige Ablaufbeispiele für die Datei *suchtext.txt* angegeben. Die Datei *suchtext.txt* habe hierbei den folgenden Inhalt:

```
Lieber Herr Meier

ich hoffe, dass Sie den vorgegebenen Termin einhalten koennen.
Sollten Sie aus irgendwelchen Gruenden in Verzug kommen,
so lassen Sie uns das rechtzeitig wissen, damit wir uns
auf diese Verzoegerung einstellen konnten.
Auch sollten Sie, Herr Meier, uns bei irgendwelchen
Unklarheiten sofort kontaktieren, damit wir diese
rechtzeitig klaeren koennen.

Mit freundlichen Gruessen
-- Die Projektleiter
```

Für diese Datei könnten sich z.B. die folgenden Abläufe ergeben:

```
user@linux: > suchtext Meier suchtext.txt ↵
1: Lieber Herr Meier
7: Auch sollten Sie, Herr Meier, uns bei irgendwelchen
user@linux: > _
```

```
user@linux: > suchtext Meier suchtext.tex ↵
...kann Datei 'suchtext.tex' nicht oeffnen
user@linux: > _
```

```
user@linux: > suchtext oll suchtext.txt ↵
4: Sollten Sie aus irgendwelchen Gruenden in Verzug kommen,
7: Auch sollten Sie, Herr Meier, uns bei irgendwelchen
user@linux: > _
```

```
user@linux: > suchtext Herr Meier suchtext.txt ↵
usage: suchtext string datei
user@linux: > _
```

```
user@linux: > suchtext Sie suchtext.txt ↵
3: ich hoffe, dass Sie den vorgegebenen Termin einhalten koennen.
4: Sollten Sie aus irgendwelchen Gruenden in Verzug kommen,
5: so lassen Sie uns das rechtzeitig wissen, damit wir uns
7: Auch sollten Sie, Herr Meier, uns bei irgendwelchen
user@linux: > _
```

```
user@linux: > suchtext Hallo suchtext.txt ↵ [wenn nicht vorhanden->keine Ausgabe]
user@linux: > _
```

### 30.1.5 Ausgeben bestimmter Zeilen einer Datei *Allgemein(6;<100)*

Erstellen Sie ein Programm *zeilausg.c*, das aus einer Datei nur bestimmte Zeilen ausgibt. Welche Zeilen auszugeben sind, soll dabei auf der Kommandozeile angegeben werden, wie z.B.

#### **zeilausg 2-10 text.txt**

Die Zeilen 2 bis 10 von der Datei *text.txt* ausgeben.

#### **zeilausg 3,4-9,12,14- gebuehr.txt**

Zeilen 3, 4 bis 9, 12 und ab Zeile 14 alle Zeilen der Datei *gebuehr.txt* ausgeben.

#### **zeilausg -20,50- kunden.txt**

Von der Datei *kunden.txt* die ersten 20 Zeilen und ab Zeile 50 alle Zeilen bis zum Dateiende ausgeben.

#### **zeilausg maerchen.txt**

Die Datei *maerchen.txt* vollständig ausgeben.

## 30.2 Elementare E/A-Funktionen

### 30.2.1 Anhängen einer Datei an eine andere *Allgemein(3;<40)*

Erstellen Sie ein Programm *anhaeng.c*, das zwei Dateinamen auf der Kommandozeile erwartet und dann unter Verwendung der elementaren E/A-Funktionen den Inhalt der zuerst angegebenen Datei an die zweite Datei anhängt.

### 30.2.2 Rückwärtiges Ausgeben einer Datei *Allgemein(5;<60)*

Erstellen Sie ein Programm *reverse.c*, das unter Verwendung der elementaren E/A-Funktionen eine Datei, deren Name auf der Kommandozeile anzugeben ist, Zeile für Zeile rückwärts ausgibt.

# Index

- adrdruck.c, 92
- adturing.c, 40
- anhaeng.c, 94
  
- basket.c, 11
- billard.c, 3
- bin\_op.c, 24
- binarray.c, 82
- binom.c, 27
- block.c, 35
- bucstrei.c, 55
  
- c\_worte.c, 72
- consfunk.c, 39
- constzei.c, 39
  
- datediff.c, 76
- dhondt.c, 62
- doppsort.c, 79
- doturing.c, 40
- dreidim.c, 7
- dualaddi.c, 40
- dualrech.c, 66
- dualwand.c, 28
  
- farben.c, 64
- float.c, 41
  
- gauselim.c, 53
- ggtrekur.c, 28
  
- hauskauf.c, 69
- heirat.c, 84
- huhn.c, 15
  
- jahrtag.c, 60
  
- kaefer.c, 58
- katzmaus.c, 9
- koch.c, 16
- kommaadd.c, 71
- kreis.c, 12
  
- lastline.c, 91
- laugsaeu.c, 89
- life.c, 49
- limits.c, 41
- linkreis.c, 13
- listmisc.c, 77
- lotto.c, 43
  
- male.c, 21
- mandel.c, 17
- matmult.c, 46
- modulb.c, 38
- modulc.c, 38
- modus.c, 54
- morse.c, 72
- multipli.c, 40
- muturing.c, 40
  
- noteverw.c, 74
- nulleins.c, 54
- numausg.c, 92
- numrueck.c, 67
  
- palindro.c, 56
- piregen.c, 4
- primsieb.c, 44
- primza.c, 68
- pythbaum.c, 31
  
- quersum.c, 29
  
- reverse.c, 94
- romzahl.c, 73
- romzahl2.c, 73
- rotkreis.c, 12
  
- sort4zah.c, 25
- speikla1.c, 36
- speikla2.c, 37
- speikla3.c, 38
- spiro.c, 5
- strekzug.c, 23
- strilist.c, 67
- suchtext.c, 93
  
- terasse.c, 30
- tictac.c, 48
  
- uhr.c, 6
- uhr3d.c, 6
  
- verdoppl.c, 40
- vielmax.c, 27
  
- wochetag.c, 59
- wortadd.c, 64
- wortlen.c, 57
- wortrepl.c, 55
- wortstat.c, 81
- wurmelist.c, 14
  
- zahlein.c, 56
- zahlfolg.c, 61
- zahlsys.c, 65
- zeilausg.c, 94
- zeitadd.c, 76
- zeitrech.c, 25
- ziegprob.c, 45