

Kapitel 2

Elementare Operationen

2.1 Teilbarkeit durch Subtraktionen

Schreiben Sie ein Programm, dass die Teilbarkeit zweier natürlicher Zahlen größer 0 prüft. Vergeben Sie für die Zahlen den Typ **unsigned** und verwenden Sie ausschließlich die Operation "Subtraktion", um die Aufgabe zu lösen. Beispiel:

```
Geben Sie die Zahlen ein:  
p = 45  
q = 9  
45 ist durch 9 teilbar.
```

Problemanalyse und Entwurf der Lösung

Nach dem Einlesen der beiden Zahlen bestimmen wir mit sukzessiven Subtraktionen, ob p durch q teilbar ist: Solange p größer gleich q ist, ziehen wir q von p ab. Es gilt, dass p dann und nur dann durch q teilbar ist, wenn p durch die fortschreitenden Subtraktionen gleich 0 wird. Im Programm initialisieren wir $p1$ mit p , weil wir die zuerst eingegebene Zahl für die Ausgabe benötigen.

Programm

```
#include <stdio.h>  
  
int main(void) {  
    unsigned p, q, p1;  
    printf("Geben Sie die Zahlen ein: \np = ");  
    scanf("%d", &p); p1 = p;  
    printf("q = ");  
    scanf("%d", &q);  
    while (p >= q)  
        p -= q;  
    if (p!=0) printf("%d ist durch %d teilbar.\n", p1, q);  
    else printf("%d ist nicht durch %d teilbar.\n", p1, q);  
}
```

```
    return 0;
}
```

Aufgaben

1. Ändern Sie das Programm so, dass man beliebige ganze Zahlen eingeben kann.
2. Erweitern Sie das Programm so, dass auch der Quotient und der Rest der Division von p durch q angezeigt werden.

2.2 Euklidischer Algorithmus

Schreiben Sie ein Programm, das den größten gemeinsamen Teiler zweier Zahlen mit Hilfe des Euklidischen Algorithmus berechnet. Es wird angenommen, dass die gegebenen natürlichen Zahlen in den Typ `unsigned long` passen. Beispiel:

```
a = 73356
b = 326
-----
ggT(73356, 326) = 2
```

Problemanalyse und Entwurf der Lösung

Der Euklidische Algorithmus zur Berechnung des größten gemeinsamen Teilers zweier natürlichen Zahlen: Seien $a, b \in \mathbb{N}$ mit $a \geq b$ und $a_1 = a$ und $b_1 = b$. Wir definieren die Paare (m_i, r_i) , so dass $a_i = m_i b_i + r_i$ mit $0 \leq r_i < b_i$. Für einen beliebigen Index i sei außerdem $a_i + 1 = b_i$ und $b_i + 1 = r_i$. Dann gibt es einen Index k , so dass $r_k = 0$ ist. Für dieses k gilt $\text{ggT}(a, b) = r_{k-1}$ (größter gemeinsamer Teiler von a und b). Der Algorithmus in Pseudocode:

ALGORITHM_EUKLID

1. Lese $a, b \in \mathbb{N}$, $a \geq b > 0$
2. $a_1 \leftarrow a$, $b_1 \leftarrow b$, $i \leftarrow 1$
3. **While** ($b_i \neq 0$) **Do**
 - 3.1 $a_{i+1} \leftarrow b_i$
 - 3.2 $b_{i+1} \leftarrow r_i (= a_i \bmod b_i)$
 - 3.3 $i \leftarrow i + 1$

End_While

4. $\text{ggT}(a, b) = r_{i-1}$

END_ALGORITHM_EUKLID

Abb. 2.1: Pseudocode für Euklidischen Algorithmus

Im Programm bilden wir die Schritte des Algorithmus so ab: Solange b ungleich 0 ist, erhält b den Wert $(a \bmod b)$ und a den Wert des alten b . In zusätzlichen Variablen merken wir uns die ursprünglichen Werte a und b , weil sie sich ändern.

Programm

```
#include <stdio.h>

int main(void){
    unsigned long a, b, r;
    unsigned long auxa, auxb;
    printf( "a = " ); scanf( "%lu", &a );
    printf( "b = " ); scanf( "%lu", &b );
    auxa = a; auxb = b;
    while(b != 0){
        r = a % b;
        a = b;
        b = r;
    }
    printf("-----\n");
    printf( "ggT(%lu, %lu) = %lu", auxa, auxb, a );
    return 0;
}
```

Aufgaben

1. Was passiert, wenn die **while**-Schleife im Programm nur so aussieht?

```
while (b != 0) {
    a = b;
    b = r;}
```

2. Schreiben Sie ein Programm, das den größten gemeinsamen Teiler mit diesem Algorithmus berechnet:

ALGORITHM_GGT

1. Lese $a, b \in \mathbb{N}$, $a \geq b > 0$

2. **While** ($b_i \neq 0$) **Do**

If ($a > b$)

$a \leftarrow a - b$

Else

$b \leftarrow b - a$

End_While

3. $\text{ggT}(a, b) = a$

END_ALGORITHM_GGT

Abb. 2.2: Pseudocode für einen GGT-Algorithmus

2.3 Einfacher Primalitätstest

Entwerfen Sie ein Programm, das herausfindet, ob eine natürliche Zahl prim ist. Wir nehmen an, dass die Zahl in den Datentyp **unsigned long** passt. Falls die Zahl nicht prim ist, geben Sie ihren kleinsten Teiler größer 1 aus, wie im Beispiel:

<pre>n = 35347 35347 ist nicht prim! Kleinster Teiler groesser 1 ist 13</pre>	<pre>n = 7919 7919 ist eine Primzahl!</pre>
-------------------------------------------------------------------------------	---------------------------------------------

Problemanalyse und Entwurf der Lösung

Wir verwenden die Variable k , die schrittweise alle natürlichen Zahlen zwischen 2 und \sqrt{n} aufnimmt, solange kein Teiler von n gefunden wird. Das wird in einer **while**-Schleife erledigt. Zu Beginn nehmen wir an, dass die Zahl prim ist, deshalb setzen wir die Variable `prim` auf 1. Wenn ein Teiler von n ermittelt wird, weisen wir `prim` den Wert 0 zu.

Programm

```
#include <stdio.h>

int main(void) {
    unsigned long n, k;
    short prim = 1;
    printf("n = "); scanf("%lu", &n);
    k = 2;
    while(prim && k*k<=n) {
        prim = (n%k != 0);
        k++;
    }
    if(prim) printf( "\n%lu ist eine Primzahl!", n );
    else {
        printf("\n%lu ist nicht prim!", n);
        printf("\nKleinster Teiler groesser 1 ist  %lu", --k);
    }
    return 0;
}
```

Aufgaben

1. Schauen Sie sich die Lösung des sechsten Problems im ersten Kapitel an.
2. Verwenden Sie statt der **while**- eine **do-while**-Schleife.
3. Was passiert, wenn Sie die ganze **while**-Schleife im Programm inklusive ihres Körpers durch die folgende einzelne Zeile ersetzen?

```
while ((prim=(n%k++!=0)?1:0) && k<sqrt(n));
```

Erklären Sie das Ergebnis.

4. Es ist ausreichend, wenn wir k nur mit 2 und den ungeraden natürlichen Zahlen bis \sqrt{n} belegen. Implementieren Sie die Verbesserung.
5. Schreiben Sie ein Programm, das alle Primzahlen mit 4 Ziffern ausgibt.

2.4 Der Punkt mit dem kürzesten Abstand

Es sei der Punkt P_0 und eine Menge A mit n Punkten in der Ebene gegeben. Schreiben Sie ein Programm, das den Punkt findet, der P_0 am nächsten liegt und den Abstand zwischen diesen beiden Punkten ausgibt. Die Punkte mit ihren Koordinaten geben wir über die Tastatur ein.

Beispiel

```
Geben Sie P0 ein:
x = 5.43
y = 3.21
Geben Sie die Anzahl der Punkte in A ein: 3
Geben Sie die Koordinaten ein:
Erster Punkt:
x = 1.2
y = 2.43
2-ter Punkt:
x = 5.43
y = 1.2
3-ter Punkt:
x = 6.54
y = 3.21
Der Punkt mit dem kleinsten Abstand zu P0 ist (6.54, 3.21)
Der Abstand betraegt 1.1100
```

Problemanalyse und Entwurf der Lösung

Es seien zwei Punkte $P_1(x_1, y_1)$ und $P_2(x_2, y_2)$ in der Ebene gegeben. Dann ist der Abstand zwischen ihnen:

$$P_1P_2 = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Wir stellen zwei Programme vor, die die Aufgabe lösen. Das erste arbeitet ohne und das zweite mit benutzerdefinierten Datentypen und Funktionen.

Im ersten Programm lesen wir den Punkt P_0 ein und dann die Punkte aus der Menge A . Die Variable `dist` enthält den kleinsten Abstand, der zwischen P_0 und den bisher eingelesenen Punkten ermittelt wurde.

Im zweiten Programm verwenden wir den Datentyp `TPoint`, um die Punkte darzustellen und außerdem die Funktionen

```
double dist(TPoint p, TPoint q);
```

und

```
int readPoint(TPunct *p);
```

um den Abstand zwischen zwei Punkten zu berechnen und einen Punkt zu lesen. Beachten Sie den Adressoperator `*` für den Parameter der Funktion `readPoint()`.

Erstes Programm

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

int main(void) {

    double x0, y0, x1, y1, x, y;
    double dist, d;
    int n, i;
    printf("Geben Sie P0 ein: \nx = ");
    scanf("%lf", &x0);
    printf("y = ");
    scanf("%lf", &y0);
    printf("Geben Sie die Anzahl der Punkte in A ein: ");
    scanf("%d", &n);
    printf("Geben Sie die Koordinaten ein: ");
    printf("Erster Punkt: \nx = ");
    scanf("%lf", &x); x1 = x;
    printf("y = "); scanf("%lf", &y); y1 = y;
    dist = sqrt((x0 - x)*(x0 - x) + (y0 - y)*(y0 - y));

    for(i = 2; i <= n; i++) {
        printf("\n%d-ter Punkt: \nx = ", i);
        scanf("%lf", &x);
        printf("y = ");
        scanf("%lf", &y);
        d = sqrt((x0 - x)*(x0 - x) + (y0 - y)*(y0 - y));
        if (d < dist) {
            x1 = x;
            y1 = y;
            dist = d;
        }
    }
    printf(
```

```

    "\nDer Punkt mit dem kleinsten Abstand zu P0 ist      \
    (%.2lf, %.2lf) \nDer Abstand betraegt %.4lf", x1, y1, dist);
    return 0;
}

```

Zweites Programm

```

#include <stdio.h>
#include <math.h>

typedef struct{
    double x, y;
}TPoint;

double dist(TPoint p, TPoint q){
    return
        sqrt((p.x - q.x)*(p.x - q.x) + (p.y - q.y)*(p.y - q.y));
}

int readPoint(TPoint *p){
    printf("x = "); scanf("%lf", &p->x);
    printf("y = "); scanf("%lf", &p->y);
    return 0;
}

int main(void){

    int n, i;
    TPoint P0, P1, P;
    double dAux, d;
    printf("Der Punkt P0: \n"); readPoint( &P0 );
    printf("Geben Sie die Anzahl der Punkte in A ein: ");
    scanf("%d", &n);
    printf("Geben Sie die Koordinaten ein: \n");
    printf("Erster Punkt: \n"); readPoint( &P1 );
    d = dist( P0, P1 );

    for(i=2; i<=n; i++){
        printf("\nDer %d-te Punkt:\n", i);
        readPoint(&P);
        dAux = dist(P0, P);
        if(dAux < d){
            P1 = P;
            d = dAux;
        }
    }
    printf(
        "\nDer Punkt mit dem kleinsten Abstand zu P0 ist      \
        (%.2lf, %.2lf) \nDer Abstand betraegt %.4lf", P1.x, P1.y, d)
        ;
    return 0;
}

```

Aufgaben

1. Machen Sie ein Beispiel auf einem Blatt Papier mit P_0 und vier weiteren Punkten. Schreiben Sie auch die Schritte des Algorithmus nieder.
2. Ändern Sie das zweite Programm so ab, dass die Ermittlung der Distanzen erst dann erfolgt, wenn alle Punkte eingelesen wurden. Nutzen Sie dafür ein eindimensionales Array mit Elementen vom Typ `TPoint`.
3. Ergänzen Sie beide Programme um die Ausgabe des Punktes, der am weitesten von P_0 entfernt ist und des Abstandes zwischen diesen Punkten.
4. Gegeben sind die Punkte $P_1(x_1, y_1)$ und $P_2(x_2, y_2)$. Die Koordinaten des Mittelpunktes $M(x, y)$ der Strecke P_1P_2 sind:

$$x = \frac{x_1 + x_2}{2}, \quad y = \frac{y_1 + y_2}{2}$$

Erweitern Sie beide Programme so, dass sie die Mittelpunkte der Strecken zwischen P_0 und allen Punkten der Menge A ausgeben.

5. Es sei eine Menge A mit n Punkten in der Ebene gegeben. Schreiben Sie ein Programm, das den am weitesten von der x - und y -Achse entfernten Punkt ausgibt.

2.5 Größe des Speicherplatzes

Innerhalb einer Funktion darf eine beliebige Anzahl von Variablen verwendet werden. Ermitteln Sie, wieviel Speicher insgesamt für die Variablen erforderlich ist. Wir nehmen an, dass dieser Wert (in Bytes) in den Datentyp `long` passt.

Beispiel:

```
Anzahl der verwendeten Datentypen? 3

Wieviel Bytes benoetigt der Datentyp 1? 200
Wieviel Variablen werden von diesem Typ gebraucht? 3
Wieviel Bytes benoetigt der Datentyp 2? 300
Wieviel Variablen werden von diesem Typ gebraucht? 7
Wieviel Bytes benoetigt der Datentyp 3? 400
Wieviel Variablen werden von diesem Typ gebraucht? 6

Gesamtgroesse des erforderlichen Speichers = 5100 Bytes.
```

Problemanalyse und Entwurf der Lösung

Wir fragen zuerst nach der Anzahl der verschiedenen Datentypen (n), die für die Variablen benötigt werden. Dann wollen wir schrittweise für jeden Typ wissen, wie

viel Bytes er im Speicher belegt (`dim`) und wie viel Variablen auf ihm deklariert werden (`nv`). Jedes mal zählen wir zur Variablen `s`, die mit 0 initialisiert ist, das Produkt aus `dim` und `nv` hinzu.

Programm

```
#include <stdio.h>

int main(void){
    int n, i;
    int nv, dim;
    long s = 0;
    printf("Anzahl der verwendeten Datentypen?");
    scanf("%d", &n);
    for(i=1; i<=n; i++){
        printf("Wieviel Bytes benoetigt der Datentyp %d?", i);
        scanf("%d", &dim);
        printf("Wieviel Variablen werden von diesem Typ gebraucht?");
        scanf("%d", &nv);
        s += nv * dim;
    }
    printf(
        "\nGesamtgroesse des erforderlichen Speichers = %ld Bytes.\n",
        s);
    return 0;
}
```

Aufgabe

Lassen Sie einen Fehler ausgeben, wenn der Typ `long` für die gesamte Anzahl der Bytes zu klein wird.

2.6 Goldener Schnitt

Leonardo Pisano, der als Fibonacci bekannt ist, wurde ca. 1180 in Pisa geboren. Weil sein Vater Handel mit nordafrikanischen Ländern betrieb, erlernte Fibonacci die hindu-arabischen Ziffern und die Rechenmethoden der arabischen Mathematiker. Fibonacci schrieb das Buch "Liber Abaci" (Buch vom Abakus), in dem er die Anwendung der hindu-arabischen Ziffern befürwortet und eindrucksvolle mathematische Probleme vorstellt, die später immer wieder von anderen Autoren aufgegriffen wurden, so z. B.:



Abb. 2.3: Fibonacci
Quelle im Anhang B

Jemand setzt ein Kaninchenpaar in einen Garten, der von einer Mauer umgeben ist. Wie viele Kaninchenpaare werden jedes Jahr geboren, wenn man annimmt, dass jeden Monat jedes Paar ein weiteres Paar zeugt, und dass Kaninchen ab dem Alter von zwei Monaten geschlechtsreif sind?

Die Zahlenreihe, die aus diesem Problem abgeleitet werden kann, ist als Fibonacci-Folge bekannt: $F_0 = 0$, $F_1 = 1$ und weiter $F_{n+1} = F_n + F_{n-1}$. Die ersten Fibonacci-Zahlen sind also: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377,... Der Quotient zweier aufeinander folgender Fibonacci-Zahlen nähert sich an den Goldenen Schnitt an, der eine vielfältige Bedeutung in der Kunst und Natur hat.

Schreiben Sie ein Programm, das das n -te Glied der Folge $\frac{F_n}{F_{n+1}}$ ausgibt ($0 \leq n \leq 40$).

Beispiel

```
Geben Sie den Index ein: 4
F4/F5 = 0.6250
```

Problemanalyse und Entwurf der Lösung

Innerhalb einer **while**-Schleife berechnen wir iterativ die Fibonacci-Zahlen, wobei **f** und **fNew** die momentan letzten beiden Glieder widerspiegeln.

Programm

```
#include <stdio.h>

int main(void) {

    long f = 1, fNew = 1;
    long fAux;
    int n, naux;
    printf("Geben Sie den Index ein: ");
    scanf("%d", &n); naux = n;

    while(naux) {
        naux--;
        fAux = f + fNew;
        f = fNew;
        fNew = fAux;
    }
    printf("F%d/F%d = %.4f", n, n+1, (double)f/fNew);
    return 0;
}
```

Aufgaben

1. Ändern Sie das Programm so ab, dass es die n -te Fibonacci-Zahl berechnet ($0 \leq n \leq 40$). n wird wieder von der Tastatur gelesen.
2. Erweitern Sie das Programm so, dass es alle Quotienten $\frac{F_i}{F_{i+1}}$ ausgibt ($0 \leq i \leq 40$), einen Quotienten pro Zeile. Was bemerken Sie?
3. Lesen Sie bei Wikipedia über die Fibonacci-Folge und den Goldener Schnitt nach:
<http://de.wikipedia.org/wiki/Fibonacci-Folge>
http://de.wikipedia.org/wiki/Goldener_Schnitt

2.7
Position eines Punktes im Kreis

Fünf reelle Zahlen werden von der Tastatur eingelesen: x_m , y_m , r , x_0 und y_0 . Sie beschreiben den Mittelpunkt $M(x_m, y_m)$ und Radius (r) eines Kreises und die Koordinaten (x_0, y_0) eines Punktes P in der Ebene. Entwickeln Sie ein Programm, das entscheidet, ob der Punkt innerhalb oder außerhalb des Kreises oder auf der Kreislinie liegt.

Beispiel

Variante 1	Variante 2
Geben Sie den Kreis ein: Mittelpunkt: xm = 0 ym = 0 Radius = 2 Geben Sie den Punkt P ein: x0 = 1 y0 = 1 P ist innerhalb des Kreises	Geben Sie den Kreis ein: Mittelpunkt: xm = -67 ym = -45.71 Radius = 34.56 Geben Sie den Punkt P ein: x0 = 1000.78 y0 = 678.90 P ist ausserhalb des Kreises

Problemanalyse und Entwurf der Lösung

Die Gleichung des Kreises lautet:

$$(x - x_m)^2 + (y - y_m)^2 = r^2$$

Alle Punkte in der Ebene, die diese Gleichung erfüllen, befinden sich auf dem Kreisrand. Die Punkte in der Ebene, die die Ungleichung

$$(x - x_m)^2 + (y - y_m)^2 < r^2$$

erfüllen, befinden sich innerhalb des Kreises. Die Punkte in der Ebene, die die Ungleichung

$$(x - x_m)^2 + (y - y_m)^2 > r^2$$

erfüllen, befinden sich außerhalb des Kreises. Das folgende Programm basiert auf diesen Feststellungen. Wir berechnen den Abstand d zwischen dem Mittelpunkt $M(x_m, y_m)$ und dem Punkt $P(x_0, y_0)$:

$$MP = \sqrt{(x_0 - x_m)^2 + (y_0 - y_m)^2}$$

Programm

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

int main(void) {
    float xm, ym, r, x0, y0;
    double d;
    printf("Geben Sie den Kreis ein:\n");
    printf("Mittelpunkt:\n x = "); scanf("%f", &xm);
    printf(" y = "); scanf("%f", &ym);
    printf("Radius = "); scanf("%f", &r);
    printf("Geben Sie den Punkt P ein:\n");
    printf(" x0 = "); scanf("%f", &x0);
    printf(" y0 = "); scanf("%f", &y0);
    d = sqrt((x0-xm)*(x0-xm)+(y0-ym)*(y0-ym));
    if (d>r) printf("P ist ausserhalb des Kreises");
    else if (d==r)
        printf("P ist auf der Kreislinie");
    else
        printf("P ist innerhalb des Kreises");
    return 0;
}
```

Aufgaben

1. Modifizieren Sie das Programm so, dass es mehrere Kreise bei der Eingabe akzeptiert und feststellt, ob der Punkt innerhalb der Schnittfläche aller Kreise liegt.
2. Schreiben Sie ein Programm, das einen Punkt und mehrere Quadrate, beschrieben durch ihren oberen linken Punkt und ihre Kantenlänge, einliest. Das Programm soll ausgeben, ob der Punkt innerhalb der Schnittfläche aller Quadrate liegt.
3. Es seien a und c reelle positive Zahlen mit $a > c$ und F_1 und F_2 feste Punkte in der Ebene mit der Eigenschaft $F_1 F_2 = 2c$. Die Menge aller Punkte P in der Ebene mit der Eigenschaft $PF_1 + PF_2 = 2a$ heißt *Ellipse* (siehe Abb. 2.4).

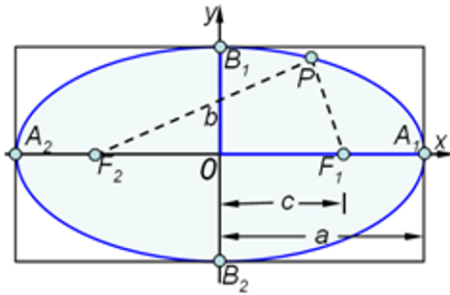


Abb. 2.4: Ellipse

Ein Punkt $P(x,y)$ gehört dann und nur dann der Ellipse E an, wenn:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1 \quad \text{wobei} \quad b = \sqrt{a^2 - c^2}$$

Implementieren Sie ein Programm, das entscheidet, ob sich ein Punkt innerhalb oder außerhalb einer Ellipse befindet, oder auf ihrem Rand.

2.8 Das arithmetische Mittel

Schreiben Sie ein Programm, das das arithmetische Mittel n reeller Zahlen berechnet, die von der Tastatur eingelesen werden. Für die Zahlen genügt der Datentyp `float` und n ist kleiner als die Konstante `MAX_INT`. Das Programm darf keine Arrays verwenden. Beispiel:

```
Wie viele Zahlen sind es? 7
Geben Sie die Zahlen ein:
7 1 2 3 4 7.89 4.53
Das arithmetische Mittel: 4.203.
```

Problemanalyse und Entwurf der Lösung

Das arithmetische Mittel der Zahlen a_1, a_2, \dots, a_n berechnet man mit der Formel:

$$m_a = \frac{a_1 + a_2 + \dots + a_n}{n} = \frac{a_1}{n} + \frac{a_2}{n} + \dots + \frac{a_n}{n}$$

Weil es sein kann, dass die Summe der Elemente in keinen Datentyp passt, arbeiten wir mit dem rechten Teil der Gleichung.

Programm

```
#include <stdio.h>

int main(void) {
```

```

float a;
int n, i;
float mA = 0;
printf("Wie viele Zahlen sind es?");
scanf("%d", &n);
printf("Geben Sie die Zahlen ein: ");
for(i=0; i<n; i++){
    scanf("%f", &a);
    mA += a/n;
}
printf("Das arithmetische Mittel: %.3f.\n", mA);
return 0;
}

```

Aufgaben

1. Erweitern Sie das Programm so, dass es den Wert

$$m_b = \sqrt{\frac{a_1^2 + a_2^2 + \dots + a_n^2}{n}}$$

berechnet, wobei die a_i^2 in den Typ **float** passen und die Summe $a_1^2 + a_2^2 + \dots + a_n^2$ beliebig groß sein kann.

2. Schreiben Sie ein Programm, das n reelle Zufallszahlen generiert, mit ihnen die Werte m_a und m_b berechnet und die Ungleichung

$$m_a \leq m_b$$

prüft. Was stellen Sie fest, wenn Sie das Programm mehrere Male laufen lassen? Beweisen Sie Ihre Beobachtung mathematisch.

2.9 Lineare Rekurrenz

Schreiben Sie ein Programm, das das i -te Mitglied der Folge

$$f(0) = 0, \quad f(n) = \frac{f(n-1) + n}{2}$$

bestimmt. Beispiel:

```

n = 5
f(5) = 4.031

```

Problemanalyse und Entwurf der Lösung

Wir berechnen innerhalb einer **for**-Schleife schrittweise die Elemente der Folge.

Programm

```
#include <stdio.h>

int main(void) {
    float f;
    int n, i;
    printf("n = ");
    scanf("%d", &n);
    f=0;
    for(i=1; i<=n; i++)
        f = (f + i)/2;
    printf("f(%d) = %.3f\n", n, f);
    return 0;
}
```

Aufgaben

1. Verwenden Sie anstatt der **for**- eine **while**-Schleife.
2. Ändern Sie das Programm so, dass es alle Werte $f(i)$ ausgibt ($1 \leq i \leq 100$). Was bemerken Sie? Beweisen Sie das Resultat (eventuell durch vollständige Induktion).

2.10 Synonyme Funktion mit `atoi()`

In der Headerdatei `stdlib.h` befinden sich mehrere Funktionen für die numerische Umwandlung. Ein Beispiel ist die Funktion `atoi()`, die eine Zeichenkette in ihr numerisches Äquivalent vom Datentyp `long` umwandelt. Entwerfen Sie eine Funktion, die dieselbe Aufgabe erfüllt wie die Funktion `atoi()`. Beispiele:

```
Geben Sie die Zeichenkette ein: 56789012
atoi(56789012) = 56789012.
```

```
Geben Sie die Zeichenkette ein: 56tg678
Die Eingabe ist keine Zahl!
```

Problemanalyse und Entwurf der Lösung

Wir schreiben die Funktion `isNumber()`, die entscheidet, ob die gegebene Zeichenkette eine gültige ganze Zahl ist: Zuerst wird mit einer **if**-Anweisung geprüft, ob

das erste Zeichen "+" oder "-" ist. Dann wird für jedes weitere Zeichen der Zeichenkette festgestellt, ob es eine Ziffer ist. Sehen Sie sich an, wie wir dafür die Funktion `isdigit()` aus der Bibliothek `ctype.h` einsetzen. `isNumber()` liefert 1 zurück, wenn die Zeichenkette eine gültige Zahl ist, und 0, wenn nicht.

Wir nennen unsere selbst implementierte Funktion auch `atol()`. Wieder prüfen wir zuerst, ob die Zeichenkette mit einem "+" oder "-" beginnt. Die Variable `sign` initialisieren wir mit 1, und wenn das erste Zeichen "-" lautet, setzen wir `sign` auf -1. Nun lesen wir eine Ziffer `s[i]` nach der anderen aus der Zeichenkette, multiplizieren die Variable `result`, die mit 0 initialisiert ist, mit 10 und addieren `s[i]`. Wenn zum Beispiel `result` den Wert 345 enthält, und sich im aktuellen `s[i]` die Ziffer 9 befindet, wird das neue `result = 345*10+9`, also 3459. Weil die Ziffern "0", "1", ..., "9" die ASCII-Codes 48, 49, ..., 57 haben, müssen wir vom aktuellen `s[i]` erst die "0", sprich den ASCII-Code 48, abziehen.

Programm

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

long atol(char[30]);
short isNumber(char[30]);

int main(void){
    char s[30];
    printf("Die Zeichenkette: ");
    scanf("%s", s);
    if(!isNumber(s))
        printf( "Die Eingabe ist keine Zahl!"  );
    else
        printf( "atol(%s) = %ld.", s, atol(s));
    return 0;
}

short isNumber(char s[30]){
    short ok = 1;
    unsigned i;
    if( !isdigit(s[0]) && s[0] != '-' && s[0] != '+' )
        ok=0;
    for( i=1; ok && i<strlen(s); i++ )
        if(!isdigit(s[i])) ok = 0;
    return ok;
}

long atol(char s[30]){
    int sign = 1;
    unsigned i = 0;
    long result = 0;
    if(s[0] == '+') i++;
    else if( s[0] == '-' ) {i++; sign = -1;}
    while (i < strlen(s)){
```



```
    result = result * 10 + (s[i] - '0');  
    i++;  
}  
result *= sign;  
return result;  
}
```

Aufgaben

1. Lassen Sie die Funktion `isNumber()` prüfen, ob die Zahl, die durch eine Zeichenkette repräsentiert ist, in den Datentyp `long` passt.
2. Schreiben Sie ein Programm, das die umgekehrte Funktion `ltoa()` erledigt, also eine Zahl in eine Zeichenkette transformiert.
3. Informieren Sie sich in der Hilfe über die Bedeutung der numerischen Funktionen `atol()`, `ltoa()`, `atoi()`, `itoa()` und `atof()` aus der Bibliothek `stdlib`.
h. Schreiben Sie ein Beispielprogramm mit allen Funktionen.

2.11 Informationen über Zeichen

Schreiben Sie ein Programm, das ein Zeichen von der Tastatur in eine Variable liest und das Zeichen selbst, seinen ASCII-Code und die Speicheradresse der Variable ausgibt. Beispiel:

```
Geben Sie das Zeichen ein: G  
Wert: G  
ASCII-Code: 71  
Adresse: 0013FED7
```

Problemanalyse und Entwurf der Lösung

Um die Informationen auszugeben, verwenden wir Formatelemente und den Adressoperator `&`.

Programm

```
#include <stdio.h>  
int main(void) {  
    char c;  
    printf("Geben Sie das Zeichen ein: ");  
    scanf("%c", &c);  
    printf(" Wert: %c\n", c);  
    printf(" ASCII-Code: %d\n", c);  
    printf(" Adresse: %p ", &c);  
    return 0;  
}
```

Aufgaben

1. Schreiben Sie ein Programm, das eine Tabelle mit allen Zeichen des ASCII-Codes erstellt: Zeichen und ASCII-Code.
2. Lesen Sie in der Hilfe nach, welche Funktionen in der Bibliothek `ctype.h` deklariert sind, und schreiben Sie ein Programm, das alle Funktionen verwendet.

2.12 Palindrom und Quersumme

Eine natürliche Zahl ist *palindrom*, wenn sie, egal ob von links nach rechts oder von rechts nach links gelesen, denselben Wert ergibt. Beispiele: 23432, 121, 7890987. Eine natürliche Zahl, die in den Datentyp **unsigned long** passt, wird von der Tastatur eingelesen. Schreiben Sie ein Programm, das prüft, ob die Zahl *palindrom* ist und außerdem ihre Quersumme berechnet. Beispiele:

```
Geben Sie die Zahl ein: 4567654

4567654 ist palindrom!
Die Quersumme von 4567654 ist 37.
```

```
Geben Sie die Zahl ein: 4567

4567 ist nicht palindrom!
Die Quersumme von 4567 ist 22.
```

Problemanalyse und Entwurf der Lösung

Eine Zahl von rechts nach links zu lesen, heißt die Zahl umzudrehen bzw. zu spiegeln. Deshalb nennen wir unsere Funktion `mirrored()`, sie liest die Ziffern von rechts nach links und baut die gespiegelte Zahl in `m` auf:

```
while(n){
    m = m*10 + n%10;
    n /=10;
}
```

Um die Quersumme zu berechnen, implementieren wir die Funktion `sumDigits()`. Die Ziffern werden nacheinander, von rechts nach links, zur Variablen `sum` addiert.

Programm

```
#include <stdio.h>
```

```

unsigned long mirrored(unsigned long n){
    unsigned long m = 0;
    while(n){
        m = m*10 + n%10;
        n /=10;
    }
    return m;
}

short sumDigits(unsigned long n){
    short sum=0;
    while(n){
        sum += (short) (n%10);
        n /= 10;
    }
    return sum;
}

int main(void){
    unsigned long n;
    printf("Geben Sie die Zahl ein: ");
    scanf("%lu", &n);
    if(n == mirrored(n))
        printf("\n %lu ist palindrom!\n", n);
    else printf("\n %lu ist nicht palindrom!\n", n);
    printf(" Die Quersumme von %lu ist %hd. ", n, sumDigits(n));
    return 0;
}

```

Aufgaben

1. Schreiben Sie auf einem Blatt Papier die Schritte und Zwischenergebnisse für die beiden Beispiele nieder.
2. Bauen Sie das Programm so um, dass die `main()`-Funktion die einzige ist.
3. Schreiben Sie ein Programm, das eine natürliche Zahl von Typ **unsigned long** einliest und entscheidet, ohne den Modulo-Operator `%` zu verwenden, ob sie teilbar durch 2, 3, 4, 5, 9 oder 25 ist.

2.13 Unendliche Wurzel

Kalkulieren Sie den Wert des Ausdrucks:

$$\sqrt{20 + \sqrt{20 + \sqrt{20 + \sqrt{20 + \dots}}}}$$

Problemanalyse und Entwurf der Lösung

Wir bezeichnen diesen Wert mit s . Dann resultiert die Gleichung

$$20 + s = s^2$$

und die einzige positive Lösung ist 5. Wir schreiben ein Programm, ohne dieses Resultat zu benutzen, und werden auch 5 als Ergebnis erhalten.

Programm

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

int main(void) {
    double s = 20;
    while(s != sqrt( 20+s )) s = sqrt(20 + s);
    printf("Resultat = %f   !!!\n", s);
    return 0;
}
```

Aufgaben

1. Wie lautet die letzte Ziffer der Zahl 7^{7^7} ?
2. Berechnen Sie für eine eingegebene, natürliche Zahl n mit maximal 9 Ziffern die letzte Ziffer der Zahl 7^{n^n} .

2.14 Reihe mit dem Wert π

Die Zahl $\pi = 3,1415926535\dots$ gibt das Verhältnis an, das der Umfang eines beliebigen Kreises zum Kreisdurchmesser hat. π ist eine irrationale Zahl (π kann nicht als Bruch von ganzen Zahlen geschrieben werden) und hat daher unendlich viele Nachkommastellen.

Schon die alten Ägypter und Babylonier kannten π und heutzutage spielt die Kreiszahl in vielen Gebieten eine Rolle, zum Beispiel in der Wahrscheinlichkeitstheorie, der Analysis, der Zahlentheorie und in der Physik. Jedes Jahr feiert das Museum „Exploratorium“ in San Francisco am 14. März den π -Tag, weil die Zahl mit 3,14 beginnt. Schreiben Sie ein Programm, das den Wert von π mit einer Fehlerabschätzung mit Hilfe der Reihe

$$\pi = 4 \cdot \left(1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots \right)$$

berechnet. Wir bezeichnen

$$\pi(n) = 4 \sum_{i=1}^n (-1)^{i-1} \cdot \frac{1}{2i-1} = 4 \left(1 - \frac{1}{3} + \dots + (-1)^{n-1} \frac{1}{2n-1} \right)$$

Das heißt $\pi(1) = 4 \cdot 1$, $\pi(2) = 4 \left(1 - \frac{1}{3} \right)$, $\pi(3) = 4 \left(1 - \frac{1}{3} + \frac{1}{5} \right)$, ...

Die Fehlerabschätzung EPS wird von der Tastatur gelesen und wir nehmen an, dass π mit der Fehlerabschätzung berechnet ist, wenn $|\pi(n+1) - \pi(n)| < EPS$ gilt, wobei $\pi(n)$ und $\pi(n+1)$ zwei nacheinander folgende Abschätzungen sind. Beispiel:

```
EPS = 0.00000001
-----
PI mit EPS 0.00000001 = 3.141593
```

Problemanalyse und Entwurf der Lösung

Die Variablen `pi1` und `pi2` nehmen schrittweise die Werte $\pi(n)/4$ bzw. $\pi(n+1)/4$ an, wobei gilt: $n = 1, 2, 3, \dots$

Den Wert $|\pi_2 - \pi_1|$ berechnen wir mit dem bedingten Operator `?:` so:

```
pi2 > pi1 ? pi2 - pi1 : pi1 - pi2
```

`pi1` wird mit 1 und `pi2` mit $1 - 1/3$ initialisiert. Die Variable `sign` alterniert zwischen -1 und 1 und die Variable `i` durchläuft die Werte $5, 7, 9, \dots$

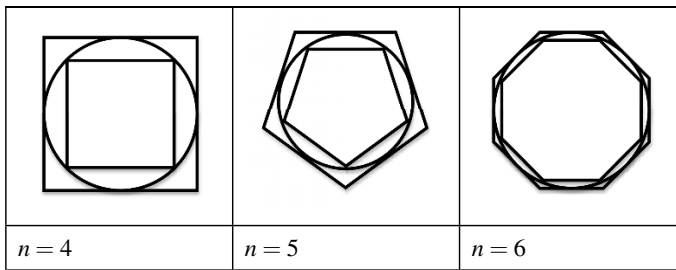
Programm

```
#include <stdio.h>

int main(void) {
    double EPS, pi1, pi2;
    int i = 5, sign = 1;
    printf("EPS = ");
    scanf("%lf", &EPS);
    pi1 = 1.0;
    pi2 = (double)1.0 - (double)1.0/3;
    while(4*(pi2 > pi1 ? pi2 - pi1 : pi1 - pi2) >= EPS) {
        pi1 = pi2;
        pi2 += sign * ((double)1.0/i);
        i += 2;
        sign *= -1;
    }
    printf("\n-----\n");
    printf("PI mit EPS %lf = %lf\n", EPS, 4*pi2);
    return 0;
}
```

Aufgaben

1. Erweitern Sie das Programm so, dass es auch die Iteration n bekannt gibt, in dem die Annäherung erreicht wird.
2. *Archimedes-Verfahren.* Archimedes von Syrakus (287-212 v. Chr.) hat als Erster eine Methode zur Bestimmung von π vorgeschlagen. Er zeichnete regelmäßige Vielecke um und in einen Kreis und hat Folgendes herausgefunden: Je mehr Ecken ein Polygon hat, desto näher kommt dessen Umfang dem Kreisumfang. Der Umfänge der Vielecke, die er um den Kreis malte, näherten sich von oben an den Kreisumfang, und die Umfänge der Vielecke, die er im Kreis konstruierte, näherten sich von unten an den Kreisumfang.



Wir geben einen Kreis mit dem Radius 1 und regelmäßige Vielecke mit $3 \times 2^{n-1}$ Kanten ($n = 1, 2, 3, \dots$) vor. Für die in dem Kreis eingezeichneten Polygone definieren wir die Folge (a_n) . Jedes Folgenglied entspricht der Hälfte des jeweiligen Polygonumfangs. Für die Vielecke, die den Kreis umschließen, definieren wir analog dazu die Folge (b_n) . Weil der Kreisumfang 2π beträgt, nähert sich (a_n) aufsteigend π an und (b_n) absteigend:

$$a_n < \pi < b_n \quad \text{für alle } n = 1, 2, \dots$$

Mit ein bisschen Trigonometrie (es gibt auch andere Wege) kommen wir zu diesen rekursiven Formeln:

$$b_{n+1} = \frac{2a_nb_n}{a_n + b_n} \quad \text{und} \quad a_{n+1} = \sqrt{a_nb_{n+1}}$$

Für die Startglieder a_0 und b_0 handelt es sich bei den Polygonen um Dreiecke, also gilt:

$$a_0 = 3 \quad \text{und} \quad b_0 = 2\sqrt{3}$$

Implementieren Sie diesen Algorithmus in einem Programm, um die genäherte Kreiszahl nach n gegebenen Iterationen auszugeben. Der Algorithmus im Pseudocode ist auf Abb. 2.5 dargestellt:

ALGORITHM_ARCHIMEDES

1. $a \leftarrow 3$

2. $b \leftarrow 2\sqrt{3}$

3. $n \leftarrow \text{Anzahl_Iterationen}$

4. **For** ($i \leftarrow 1$; $i \leq n$; step 1)

5. $b \leftarrow \frac{2ab}{a+b}$

6. $a \leftarrow \sqrt{ab}$

7. **End_For**

8. **return** a

END_ALGORITHM_ARCHIMEDES

Abb. 2.5: Archimedes-Verfahren

3. Erweitern Sie Ihr Programm aus der letzten Aufgabe. Es soll diese Tabelle erzeugen:

n	a_n	b_n
0	3	3.4641016151377544
1	3.1058285412302493	3.2153903091734728
2	3.1326286132812382	3.1596599420975005
3	3.1393502030468672	3.1460862151314348
4	3.1410319508905093	3.1427145996453683
5	3.141452472285462	3.1418730499798242

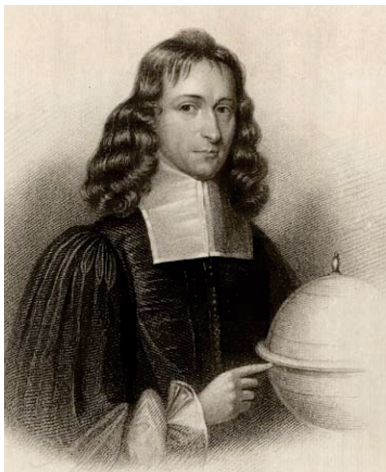
4. Aus $\tan \frac{\pi}{4} = 1$ folgt $\pi = 4 \cdot \arctan 1$.

Die Mathematiker *James Gregory* und *Gottfried Wilhelm Leibniz* haben ca. 1670 unabhängig voneinander die folgende unendliche Reihe für arctan entdeckt:

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \dots, \quad \text{für alle } x \text{ mit } x \leq 1$$

Sie kann auch so geschrieben werden:

$$\arctan x = \sum_{n=0}^{\infty} (-1)^n \left(\frac{x^{2n+1}}{2n+1} \right), \quad \text{für alle } x \text{ mit } x \leq 1$$



James Gregory (1638-1675)
Quelle im Anhang B



Gottfried Wilhelm Leibniz (1646-1716)
Quelle im Anhang B

Mit einem kleinen Testprogramm, das mit dieser Reihe den Wert $\pi = 4 \cdot \arctan 1$ berechnet, erhalten wir Werte wie:

n	$\approx \pi$
0	4
1	2.666666666666667
2	3.466666666666668
3	2.8952380952380956
4	3.3396825396825403
5	2.9760461760461765
100000	3.1416026534897203
100001	3.1415826537897158
200000	3.1415976535647618
200001	3.1415876536397613
300000	3.1415959869120198
300001	3.1415893202786864
400000	3.1415951535834941
400001	3.1415901536022441

Wir erkennen, dass die Teilfolge der geraden Glieder absteigend ist (alle Werte sind größer als π), wohingegen die Teilfolge der ungeraden Glieder aufsteigend ist (alle Werte sind kleiner als π). Der Algorithmus, der daraus folgt, ist:

ALGORITHM_ARCTAN

```

1.  $n \leftarrow \text{Anzahl\_Iterationen}$ 
2.  $x \leftarrow 1$ 
3.  $\text{Vorzeichen} \leftarrow 1$ 
4.  $\text{Zähler} \leftarrow x$ 
5.  $\text{Nenner} \leftarrow 1$ 
6. For ( $i \leftarrow 1$ ;  $i \leq n$ ; step 1)
7.  $\text{Resultat} \leftarrow \text{Vorzeichen} \cdot \frac{\text{Zähler}}{\text{Nenner}}$ 
8.  $\text{Zähler} \leftarrow \text{Zähler} \cdot x^2$ 
9.  $\text{Nenner} \leftarrow \text{Nenner} + 2$ 
10.  $\text{Vorzeichen} \leftarrow \text{Vorzeichen} \cdot (-1)$ 
11. End_For
12. return  $4.0 \cdot \text{Resultat}$ 

```

END_ALGORITHM_ARCTAN

Schreiben Sie ein auf diesem Algorithmus basierendes Programm, das die Abschätzung von π für eine eingetragene Iteration n ausgibt.

5. Eine andere bekannte irrationale Zahl ist die Eulersche Zahl e , die nicht so alt wie π ist. Eine mögliche Näherungsformel ist

$$e = 1 + \frac{1}{1!} + \frac{1}{2!} + \frac{1}{3!} + \dots + \frac{1}{n!} + \dots$$

Schreiben Sie ein Programm, das e mit der Fehlerabschätzung EPS berechnet.

Das heißt, dass $|e(n+1) - e(n)| < EPS$, wobei $e(n)$ und $e(n+1)$ zwei aufeinander folgende Abschätzungen von e durch die obige Reihe sind. Bemerkung: Der Wert $n! = 1 \cdot 2 \cdot \dots \cdot n$ (gesprochen: n Fakultät) wächst sehr schnell!

6. Berechnen Sie wie in der vorigen Aufgabe die linken Seiten der Gleichungen mit einer gegebenen Fehlerabschätzung EPS und "prüfen" Sie sie.

a. $1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots \pm \frac{1}{n!} \mp \dots = \frac{1}{e}$

b. $1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \dots \pm \frac{1}{n} \mp \dots = \ln 2$

c. $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \dots + \frac{1}{2^n} + \dots = 2$

d. $1 - \frac{1}{2} + \frac{1}{4} - \frac{1}{8} + \dots \pm \frac{1}{2^n} \mp \dots = \frac{2}{3}$

$$e. \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \frac{1}{3 \cdot 4} + \dots + \frac{1}{n(n+1)} + \dots = 1$$

$$f. \frac{1}{1 \cdot 3} + \frac{1}{3 \cdot 5} + \frac{1}{5 \cdot 7} + \dots + \frac{1}{(2n-1)(2n+1)} + \dots = \frac{1}{2}$$

$$g. \frac{1}{1 \cdot 3} + \frac{1}{2 \cdot 4} + \frac{1}{3 \cdot 5} + \dots + \frac{1}{(n-1)(n+1)} + \dots = \frac{3}{4}$$

$$h. \frac{1}{3 \cdot 5} + \frac{1}{7 \cdot 9} + \frac{1}{11 \cdot 13} + \dots + \frac{1}{(4n-1)(4n+1)} + \dots = \frac{1}{2} - \frac{\pi}{8}$$

$$i. \frac{1}{1 \cdot 2 \cdot 3} + \frac{1}{2 \cdot 3 \cdot 4} + \dots + \frac{1}{n(n+1)(n+2)} + \dots = \frac{1}{4}$$

$$j. 1 + \frac{1}{2^2} + \frac{1}{3^2} + \frac{1}{4^2} + \dots + \frac{1}{n^2} + \dots = \frac{\pi^2}{6}$$

$$k. 1 - \frac{1}{2^2} + \frac{1}{3^2} - \frac{1}{4^2} + \dots \pm \frac{1}{n^2} \mp \dots = \frac{\pi^2}{12}$$

7. Lesen Sie die Artikel über π und e auf Wikipedia:

Kreiszahl π : <http://de.wikipedia.org/wiki/Kreiszahl>

Eulersche Zahl e : http://de.wikipedia.org/wiki/Eulersche_Zahl

2.15 Der bedingte Ausdruck ?:

Die mathematische Operation $\min(x, y)$ können wir mit dem bedingten Ausdruck so formulieren:

```
(x < y) ? x : y
```

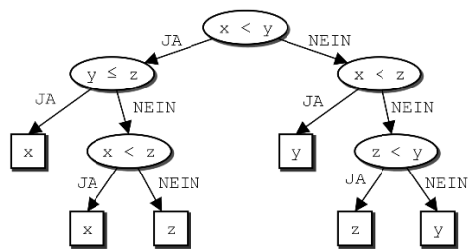
Schreiben Sie ein kurzes Programm, das den Wert $\min(x, y, z)$ damit berechnet. Beispiel:

```
Geben Sie drei Zahlen ein: 45 34 12
Minimaler Wert: 12
```

Problemanalyse und Entwurf der Lösung

Von diesem Entscheidungsbaum leiten wir die bedingte Anweisung ab:

```
x < y ? (y <= z ? x : (x < z ? x : z)) : (x < z ? y : (z < y ? z : y)) ;
```



Programm

```
#include <stdio.h>

int main(void) {
    int x, y, z;
    int min;
    printf("Geben Sie drei Zahlen ein: ");
    scanf("%d %d %d", &x, &y, &z);
    min = x<y?(y<=z?x:(x<z?x:z)):(x<z?y:(z<y?z:y));
    printf("Minimaler Wert: %d\n", min);
    return 0;
}
```

Aufgaben

- 1. Erweitern Sie das Programm so, dass es das Maximum dreier natürlicher Zahlen bestimmt.
- 2. Schreiben Sie ein Programm, das das Minimum und Maximum von vier natürlichen Zahlen ermittelt. Arbeiten Sie wieder mit bedingten Ausdrücken.

2.16 Besondere Paare

Es sei die natürliche Zahl $n, 0 < n < 9000$ gegeben. Schreiben Sie ein Programm, das

- a) alle Paare (x,y) natürlicher Zahlen ausgibt, für die $x^2 + n = y^2$ gilt,
- b) das Paar (x,y) natürlicher Zahlen findet, für das $n = 2^x \cdot (2 \cdot y + 1)$ gilt.

Beispiel:

Variante 1	Variante 2
Geben Sie n ein! n = 234	Geben Sie n ein! n = 987
a)	a)

Variante 1	Variante 2
Es gibt 0 Paare.	1. $493^2 + 987 = 494^2$
b)	2. $163^2 + 987 = 166^2$
$234 = 2^1 \cdot (2 \cdot 58 + 1)$	3. $67^2 + 987 = 74^2$
	4. $13^2 + 987 = 34^2$
	Es gibt 4 Paare.
	b)
	$987 = 2^0 \cdot (2 \cdot 493 + 1)$

Problemanalyse und Entwurf der Lösung

Die Gleichung aus **a)** formen wir so um, dass wir n als Produkt der Terme $(y - x)$ und $(y + x)$ erhalten:

$$x^2 + n = y^2 \Leftrightarrow n = y^2 - x^2 \Leftrightarrow n = (y - x) \cdot (y + x)$$

Wenn a und b Teiler von n mit den Bedingungen $a < b$ und $a \cdot b = n$ sind, können wir das einfache System

$$\begin{cases} y - x = a \\ y + x = b \end{cases}$$

aufbauen. Es hat die Lösungen:

$$x = \frac{b - a}{2} \quad y = \frac{b + a}{2}$$

Weil x und y aus \mathbb{N} sind, kommt noch die Bedingung dazu, dass $b - a$ und damit auch $b + a$ Vielfache von 2 sein müssen.

Für die Gleichung $n = 2^x \cdot (2 \cdot y + 1)$ aus **b)** gilt, dass x der Exponent von 2 in der Primfaktorzerlegung von n ist. y ist der größte ungerade Teiler von n .

Programm

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

int main(void){
    unsigned long x, y, n, nl;
    int counter = 0;
    printf("Geben Sie n ein!\n");
    printf("n = "); scanf("%lu", &n);
    printf("\na");
    for(x = 1; x <= sqrt(n); x++){
        if(n % x == 0){
            y = n / x;
            if((x + y) % 2 == 0){
                counter++;
                printf("\n %d. ", counter);
            }
        }
    }
}
```

```

        printf("%4lu^2 + %4lu = %4lu^2",
               (y-x)/2, n, (y+x)/2);
    }
}
printf("\n Es gibt %d Paare.\n", counter);
x = 0;
n1 = n;
while(n1%2 == 0){
    x++;
    n1 /= 2;
}
printf("\nb\n");
printf(" %lu = 2^%lu * (2*%lu + 1) ",
       n, x, (n1-1)/2);
return 0;
}

```

Aufgaben

1. Ändern Sie das Programm so ab, dass es bei der Bestimmung der Lösung für die Teilaufgabe **b)** zuerst y und dann x berechnet.
2. Implementieren Sie die Lösungssuche für die Teilaufgabe **a)** auf eine andere Art und Weise. Zum Beispiel mit einer *Brute-Force*-Methode, die die passenden Zahlen systematisch sucht.

2.17 Die Farey-Reihe

Eine Farey-Reihe vom Rang n besteht aus allen ausgekürzten Brüchen, deren Wert zwischen 0 und 1 liegt, und deren Nenner maximal n ist. Die Brüche sind aufsteigend sortiert.

$$F_7 = \left\{ \frac{0}{1}, \frac{1}{7}, \frac{1}{6}, \frac{1}{5}, \frac{1}{4}, \frac{2}{7}, \frac{1}{3}, \frac{2}{5}, \frac{3}{7}, \frac{1}{2}, \frac{4}{7}, \frac{3}{5}, \frac{2}{3}, \frac{5}{7}, \frac{3}{4}, \frac{4}{5}, \frac{5}{6}, \frac{6}{7}, \frac{1}{1} \right\}$$

Wenn die x_i die Zähler und die y_i die Nenner der Farey-Brüche sind, dann sieht das Bildungsgesetz einer Farey-Reihe mit dem Rang n so aus:

$$x_0 = 0, \quad x_1 = y_0 = 1, \quad y_1 = n$$

und

$$\begin{aligned} x_k &= ((x_{k-2} + n)/y_{k-1}) \cdot x_{k-1} - x_{k-2} \\ y_k &= ((y_{k-2} + n)/y_{k-1}) \cdot y_{k-1} - y_{k-2} \end{aligned}$$

Schreiben Sie ein Programm, das die Farey-Brüche für einen gegebenen Rang n ($0 \leq n \leq 1000$) ausgibt. Beispiel:

Geben Sie n ein:

$n = 7$

Farey-Brueche mit Rang 7:

0/1 1/7 1/6 1/5 1/4 2/7 1/3 2/5 3/7 1/2 4/7
3/5 2/3 5/7 3/4 4/5 5/6 6/7 1/1

Problemanalyse und Entwurf der Lösung

Analog zum Bildungsgesetz bauen wir die Reihe von links nach rechts mit den drei Variablenpaaren (x_{k-2}, y_{k-2}) , (x_{k-1}, y_{k-1}) und (x_k, y_k) auf. Weil wir uns bei dieser Lösung immer nur die letzten beiden Brüche merken wollen, initialisieren wir x_{k-2} mit 0, x_{k-1} und y_{k-2} mit 1 und y_{k-1} mit n .

Programm

```
#include <stdio.h>

int main(void){
    unsigned n, xk, yk, xk_1, yk_1, xk_2, yk_2;
    printf("Geben Sie n ein:\n n = ");
    scanf("%u", &n);
    xk_2=0; xk_1=yk_2=1; yk_1=n;
    printf(" Farey-Brueche mit Rang %u:\n", n);
    printf(" %u/%u  %u/%u ", xk_2, yk_2, xk_1, yk_1);
    while(xk_1!=1 || yk_1!=1){
        xk = (xk_2+n)/yk_1+xk_1-xk_2;
        yk = (yk_2+n)/yk_1+yk_1-yk_2;
        xk_2 = xk_1; yk_2 = yk_1;
        xk_1 = xk; yk_1 = yk;
        printf(" %u/%u ", xk, yk);
    }
    return 0;
}
```

Aufgaben

1. Schreiben Sie die Schritte des Algorithmus und die Brüche für $n = 8$ auf ein Blatt Papier.
2. Erweitern Sie das Programm so, dass es auch die Anzahl der Brüche zählt und ausgibt.
3. Finden Sie eine von n abhängige Formel für die Anzahl der Brüche.
4. Lesen Sie mehr über die Farey-Brüche im Internet nach, z. B. bei Wolfram MathWorld:

<http://mathworld.wolfram.com/FareySequence.html>

2.18 Gemeinsame Teiler

Es seien zwei natürliche Zahlen n und d gegeben, $2 < d < n$. Schreiben Sie ein Programm, das alle Paare natürlicher Zahlen (m_1, m_2) ausgibt, die beide kleiner gleich n und Vielfache von d sind. Die beiden Zahlen passen in den Typ `unsigned int`.

Beispiel:

```
Geben Sie n ein: 218
Geben Sie d ein: 37
(74, 37) (111, 37) (111, 74) (148, 37) (148, 74) (148, 111)
(185, 37) (185, 74) (185, 111) (185, 148)
```

Problemanalyse und Entwurf der Lösung

Wir stellen zwei Varianten vor. Die erste sucht mit *Brute Force* nach allen Paaren, die die Bedingungen erfüllen. Die zweite Variante durchläuft alle möglichen Vielfachen von d , indem innerhalb zweier `for`-Schleifen nur die Paare (i, j) mit der Bedingung $1 \leq j < i \leq n/d$ berücksichtigt werden. Die Lösungspaare lauten $(i \cdot d, j \cdot d)$. Das erste Programm ist natürlich viel langsamer als das zweite, prüfen Sie das mit Werten wie $n = 200.000$ und $d = 57$.

Programm Variante 1

```
#include <stdio.h>

int main(void) {

    unsigned int d, n, i, j;
    printf("Geben Sie n ein: ");
    scanf("%d", &n);
    printf("Geben Sie d ein: ");
    scanf("%d", &d);

    for(i=d; i<=n; i++)
        for(j=d; j<i; j++)
            if(j%d==0 && i%d==0)
                printf("(%d, %d)", i, j);

    return 0;
}
```

Programm Variante 2

```
#include <stdio.h>

int main(void) {

    unsigned int d, n, i, j;
```

```
printf("Geben Sie n ein: ");
scanf("%d",&n);
printf("Geben Sie d ein: ");
scanf("%d",&d);

for(i = 2; i<=n/d; i++)
    for(j=1; j<i; j++)
        printf("(%d, %d)", i*d, j*d);

return 0;
}
```

Aufgaben

- 1. Finden Sie eine von n und d abhängige Formel für die Anzahl der Paare.
- 2. Es seien die natürlichen Zahlen (m,n) gegeben. Entwickeln Sie ein Programm, das die kleinste Zahl p findet, die durch m und n teilbar ist.

2.19 Zahlenumwandlung ins Dezimalsystem

Schreiben Sie ein Programm, das mit Hilfe der Funktion `strtol()` aus der Bibliothek `stdlib.h` eine Zahl in einer gegebenen Basis ins Dezimalsystem umwandelt.

Beispiele:

Tastatur	Bildschirm
Basis: 5	1230321(Basis 5) = 23836(Basis 10)
Zahl in Basis 5: 1230321	
Basis: 2	Scannen gestoppt bei: stop
Zahl in Basis 2: 01010111111110011stop	01010111111110011(Basis 2) = 45043(Basis 10)

Problemanalyse und Entwurf der Lösung

Die Funktion `strtol()` hat den Kopf:

```
long strtol(const char *nptr, char **endptr, int basis);
```

Sie liefert eine in der Zeichenkette `nptr` gespeicherte Zahl in der Basis `basis` (zwischen 2 und 32) als Dezimalzahl vom Typ `long` zurück. Wenn das Resultat zu groß wird, gibt `strtol()` je nach Vorzeichen `LONG_MAX` oder `LONG_MIN` zurück. Eventuelle Leerzeichen am Anfang der Zeichenkette ignoriert die Funktion.

Programm

```
#include <stdio.h>
#include <stdlib.h>

int main(void) {

    int basis;
    char zahl[30], *p;
    long nr;
    printf("Basis: "); scanf("%d",&basis);
    printf("Zahl in Basis %d: ", basis);
    scanf("%s", zahl);
    nr = strtol(zahl, &p, basis);

    if(!p) printf("Ungueltiges Zeichen: %c ",*p);
    else {
        if(*p!='\0')printf("Scannen gestoppt bei: %s", p);
        *p='\0';
        printf("\n%s(Basis %d)=%ld(Basis %d)", zahl, basis, nr, 10);
    }
    return 0;
}
```

Aufgaben

1. Lesen Sie in der Hilfe mehr über die Funktion `strtol()` nach. Was passiert, wenn man als Basis 0 oder einen Wert größer 32 eingibt?
2. Informieren Sie sich auch über die verwandten Funktionen `strtod()` und `strtoul()` und schreiben Sie Beispielsprogramme damit.
3. Implementieren Sie eine eigene, zu `strtol()` synonyme Funktion.

2.20 Formatierung der natürlichen Zahlen

Schreiben Sie ein Programm, das die Zahl 12345 mit verschiedenen Formatelementen links- und rechtsbündig und mit einer bestimmten Stellenanzahl ausgibt. Beispiel:

```
12345
12345
    12345
12345
0000012345
12345
```

Problemanalyse und Entwurf der Lösung

Wir verwenden dafür die Ausgabefunktion `printf()`.

Programm

```
#include <stdio.h>

int main(void){
    int y=12345;
    printf("%d\n", y);
    printf("%5d\n", y);
    printf("%10d\n", y);
    printf("%03d\n", y);
    printf("%010d\n", y);
    printf("%-10d\n", y);
    return 0;
}
```

Aufgaben

1. Formatieren Sie auch binäre, oktale und hexadezimale Zahlen.
2. Schreiben Sie ein Testprogramm für die verschiedenen Formate für reelle Zahlen.

2.21 Vollkommene Zahlen

Eine natürliche Zahl ist vollkommen, wenn sie gleich der Summe ihrer echten positiven Teiler ist. Schreiben Sie ein Programm, das alle vollkommenen Zahlen kleiner 20.000 auflistet. Beispiel:

```
Vollkommene Zahlen kleiner 20000:
    6    28   496  8128
```

Problemanalyse und Entwurf der Lösung

Für jede Zahl `i` zwischen zwei und 20.000, berechnet man in `s` mit Hilfe einer `for`-Schleife die Summe ihrer echten positiven Teiler.

Programm

```
#include <stdio.h>
#define max 20000

int main(void){
```

```

unsigned long long i, j, s;
printf("Vollkommene Zahlen kleiner %d:\n", max);
for (i=2; i<= max; i++){
    s=0;
    for (j=1; j<=i/2; j++)
        if (i%j==0) s += j;
    if (i==s) printf("%6lu", i);
}
return 0;
}

```

Aufgaben

1. Erweitern Sie das Programm so, dass es auch alle Teiler ausgibt, und zwar in dieser Form:

```

6 = 1 + 2 + 3
28 = 1 + 2 + 4 + 7 + 14
496 = 1 + 2 + 4 + 8 + 16 + 31 + 62 + 124 + 248
8128 = 1 + 2 + 4 + 8 + 16 + 32 + 64 + 127 + 254 + 508 + 1016 +
      2032 + 4064

```

2. Hier können Sie mehr über die perfekten Zahlen erfahren:

http://de.wikipedia.org/wiki/Vollkommene_Zahl

http://www-history.mcs.st-andrews.ac.uk/HistTopics/Perfect_numbers.html

2.22 Befreundete Zahlen

Wir sagen, dass zwei natürliche Zahlen befreundet sind, wenn die Summe aller echten Teiler jeder Zahl gleich der anderen Zahl ist. Schreiben Sie ein Programm, das alle befreundeten Zahlen kleiner als 20.000 ausgibt.

Beispiel:

```

Paare befreundeter Zahlen kleiner 20000:
(284, 220) (1210, 1184) (2924, 2620) (5564, 5020) (6368,
6232) (10856, 10744) (14595, 12285) (18416, 17296)

```

Problemanalyse und Entwurf der Lösung

Zwei Varianten betrachten wir. Die erste errechnet per *Brute Force* für jede mögliche Kombination zweier natürlicher Zahlen deren Teilersummen. In der zweiten, schlauerer Variante, berechnen wir zuerst die Teilersummen aller Zahlen und speichern sie in einem Array. Daraus bedienen wir uns anschließend, wenn wir für alle

möglichen Zahlenpaare die Teilersumme jeder Zahl mit der anderen Zahl vergleichen. Weil die erste Variante die Summe der Teiler bei jedem Auftreten einer Zahl erneut berechnet, dauert die Ausführung lange, und deshalb lassen wir hier nur Zahlen kleiner 2.000 zu. Untersuchen Sie die Laufzeiten beider Programme, indem Sie die Konstante `max` mit den Werten 500, 600, ..., 2.000 belegen.

Programm Variante 1

```
#include <stdio.h>

#define max 2000

int main(void){
    unsigned long i, k, j1, j2, s1, s2;
    printf("Paare befreundeter Zahlen");
    printf(" kleiner %lu:\n", max);
    for (i=1; i<max-1; i++){
        s1=1;
        for (j1=2; j1<=i/2; j1++){
            if (i%j1==0) s1=s1+j1;
        }
        for(k=i+1; k<max; k++){
            s2=1;
            for (j2=2; j2<=k/2; j2++){
                if (k%j2==0) s2=s2+j2;
            }
            if(s2==i && s1==k)
                printf("(%lu, %lu)", i, k);
        }
    }
    return 0;
}
```

Programm Variante 2

```
#include <stdio.h>
#include <conio.h>

#define max 20000

int main(void){
    unsigned long i, j, j1, s1;
    unsigned long a[max];
    for (i=1; i<max; i++){
        s1=1;
        for(j1=2; j1<=i/2; j1++) /*Erzeugen des Tabelle A mit der
        */
            if(i%j1==0) s1=s1+j1; /* Summen aller Teiler */
        a[i-1]=s1;
    }
    printf("Paare befreundeter Zahlen");
    printf(" kleiner %lu:\n", max);
    for(i=1; i<max; i++)
        for(j=1; j<i; j++)
```

```
    if(a[i-1]==j && a[j-1]==i)
        printf("(%lu, %lu) ", i, j);
    return 0;
}
```

Aufgaben

1. Erweitern Sie beide Programme so, dass sie auch alle Teiler der Zahlen anzeigen.
2. Ermitteln Sie außerdem mit der Funktion `time()` aus der Bibliothek `time.h` die Laufzeiten beider Programme in Sekunden.

2.23 Summe der dritten Potenzen

Es sei eine natürliche Zahl n gegeben, die in den Typ `int` passt. Finden Sie alle Zahlen kleiner n , für die die Summe der dritten Potenzen der Ziffern der Zahl gleich der Zahl selbst ist.

Beispiel

```
Geben Sie n ein: 400
Ergebnis:
153
370
371
```

Problemanalyse und Entwurf der Lösung

Alle Zahlen kleiner n werden durchlaufen, und für jede wird die Summe der dritten Potenzen ihrer Ziffern bestimmt und mit der Zahl verglichen.

Programm

```
#include <stdio.h>

int main(void) {
    int n, i, j, s;
    printf("Geben Sie n ein: ");
    scanf("%d", &n);
    printf("Ergebnis:\n");
    for(i=2; i<=n; i++) {
        s=0;
        j=i;
        while(j!=0) {
            s=s+(j%10)*(j%10)*(j%10);
            j=j/10;
        }
    }
}
```

```

    if(i==s) printf(" %d\n", i);
}
return 0;
}

```

Aufgabe

Schreiben Sie ein Programm, das alle 4-Tupel (a, b, c, d) mit $a, b, c, d \in \{1, 2, \dots, 100\}$ auflistet, für die $a^3 = b^3 + c^3 + d^3$ gilt.

2.24 ASCII-Codes

Schreiben Sie ein Programm, das alle ASCII-Zeichen und deren Codes auflistet.

Beispiel

```

.....
0x1c  |||
0x1f  |▼|
0x22  |"|
0x25  |%|
0x28  |( |
0x2b  |+|
0x2e  |.|
0x31  |1|
0x34  |4|
0x37  |7|
0x3a  |:|
0x3d  |=|
0x40  |@|
0x43  |C|
0x46  |F|
0x49  |I|
0x4c  |L|
.....
0x1d  |↔|
0x20  | |
0x23  |#|
0x26  |&|
0x29  |)|
0x2c  |,|
0x2f  |/|
0x32  |2|
0x35  |5|
0x38  |8|
0x3b  |;|
0x3e  |>|
0x41  |A|
0x44  |D|
0x47  |G|
0x4a  |J|
0x4d  |M|
0x1e  |▲|
0x21  |!|
0x24  |$|
0x27  |'|
0x2a  |*|
0x2d  |-|
0x30  |0|
0x33  |3|
0x36  |6|
0x39  |9|
0x3c  |<|
0x3f  |?|
0x42  |B|
0x45  |E|
0x48  |H|
0x4b  |K|
0x4e  |N|

```

Problemanalyse und Entwurf der Lösung

Wir gehen alle Zeichen des ASCII-Codes durch und geben sie mit Formatelementen aus.

Programm

```

#include <stdio.h>

int main(void) {
    unsigned char c;

```

```

char buf[50];
for(c = 0; c < 255; c++){
    sprintf(buf, "0x%x", (int)c);
    sprintf(buf, "%s |%c|", buf, c);
    printf("%-20s", buf);
    if((c % 3) == 0) printf("\n");
}
printf("\n");
return 0;
}

```

Aufgabe

Ändern Sie das Programm so ab, dass es alle Zeichen ausgibt, die zwischen zwei von der Tastatur gelesenen Zeichen liegen.

2.25 Aufgaben

Lösen Sie alle Aufgaben, ohne Arrays zu verwenden.

1. Eine natürliche Zahl N wird eingegeben ($1 \leq N \leq 10.000$). Berechnen Sie die Anzahl der geraden und ungeraden Ziffern in N^2 .
2. Geben Sie alle Primzahlen aus, die palindrom sind und in einem gegebenen Intervall $[a, b]$ mit $1 < a < b < 10.000$ liegen.
3. Bestimmen Sie alle natürlichen palindromen Zahlen, die kleiner als eine gegebene natürliche Zahl (< 50.000) sind.
4. Ermitteln Sie alle Primzahlen mit vier Ziffern, die die Quersumme 14 haben.
5. Bestimmen Sie die maximale Zahl, die man durch das Löschen einer beliebigen Ziffer einer natürlichen Zahl N ($100 < N < 1.000.000$) erhalten kann.
6. Es sei eine natürliche Zahl N mit maximal neun Ziffern gegeben. Finden Sie die größte Zahl, die sich mit den Ziffern von N bilden lässt.
7. Wir sagen, dass eine natürliche Zahl N besonders ist, wenn eine natürliche Zahl M existiert, so dass $N = M + Q(M)$, wobei $Q(M)$ die Quersumme von M darstellt. Schreiben Sie ein Programm, das alle besonderen Zahlen aus einem Intervall $[a, b]$ ausgibt, wobei a und b natürliche Zahlen aus $[5, 50.000]$ sind.
8. Finden Sie mit einem Programm die Tripel (n, m, k) , die die Gleichung $n! + m! = k$ erfüllen, wobei k eine natürliche Zahl aus dem Intervall $[a, b]$ ist. a und b werden über die Tastatur eingegeben: $2 \leq a < b \leq 50.000$.
9. Eine natürliche Zahl in der Form $z_1 z_2 \dots z_m$ nennen wir einen Berg (ein Tal), wenn es eine Position k ($1 < k < m$) gibt, so dass die Ziffern von z_1 bis z_k aufsteigend (absteigend) und von z_k bis z_m absteigend (aufsteigend) sind. Beispiele:

136732 → Berg, 753469 → Tal. Schreiben Sie ein Programm, das eine natürliche, höchstens neunstellige Zahl einliest und danach entscheidet, ob sie Berg oder Tal ist. Erweitern Sie das Programm so, dass es alle Berge und Täler aus einem gegebenen Intervall ausgibt.

10. Es seien zwei Zahlen gegeben, sie haben maximal vier Ziffern. Berechnen Sie mit einem Programm das Produkt der beiden Zahlen mit Hilfe des klassischen arithmetischen Algorithmus. Implementieren sie die beiden Varianten:

Die amerikanische Variante, von rechts nach links	Die englische Variante, von links nach rechts
<div>981 1234 ----- 3924 2943 1962 981 ----- 1210554</div>	<div>981 1234 ----- 981 1962 2943 3924 ----- 1210554</div>

11. Wie lautet die größte Primzahl, die kleiner als 10^9 ist?
12. Bestimmen Sie die zwei kleinsten natürlichen Zahlen N mit der Eigenschaft:
- a) 7 ist ein Teiler von $N^3 + N - 2$

b) 5 ist ein Teiler von $N^3 - 3N + 6$.
13. Geben Sie alle Zahlen aus, die die Form $2^p - 2$ haben und in den Datentyp `long` passen, wobei p eine Primzahl ist. Außerdem sollen Sie für jedes p den Rest der Division von $2^p - 2$ durch p angeben. Was bemerken Sie? Können Sie es auch beweisen?
14. Wenn N eine natürliche Zahl ist, dann ist die Zahl $M = N(N + 1)(N + 2)(N + 3) + 1$ eine Quadratzahl. Prüfen Sie diese Aussage mit einem Programm für die ersten zwölf Primzahlen.
15. Entwickeln Sie ein Programm, das die letzte Ziffer von Zahlen wie 323^{700} , 1244^{51} , 1982^{83} , 164^{41} , 194^{53} , $17^{60} + 12^{40}$ und $13^{20} + 22^{30}$ bestimmt. Lassen Sie flexible Formate der Eingabedaten zu.
16. Es sei eine natürliche Zahl N gegeben, $1 \leq N \leq 5000$. Geben Sie mit einem Programm die letzte Ziffer der Zahl $1! + 2! + \dots + N!$ aus ($k! = 1 \cdot 2 \cdot \dots \cdot k$). Das Programm soll solange nach neuen N fragen und die Resultate ausgeben, bis eine Abbruchbedingung erfüllt ist (z. B. die Zahl 0 gelesen wird). Was fällt Ihnen auf? Können Sie Ihre Vermutung beweisen?
17. Finden Sie die ersten zehn Primzahlen, die die Form $4k - 1$ haben und die ersten zehn Primzahlen, die die Form $6k - 1$ haben, wobei k eine natürliche Zahl ist.

18. Es sei eine natürliche Zahl $N \leq 2000$ gegeben. Ermitteln Sie alle Primzahlen p , die kleiner als N sind und berechnen Sie für jedes p auch den Rest der Division von p^2 durch 12. Merken Sie was?
19. Wir bezeichnen das Produkt der Ziffern der natürlichen Zahl M mit $P(M)$. Berechnen Sie für eine gegebene Zahl N ($1 \leq N \leq 5000$) den Wert $P(1) + P(2) + \dots + P(N)$.
20. Wie viele Möglichkeiten gibt es, 447 als Summe von aufeinander folgenden ungeraden Zahlen darzustellen? Formulieren Sie die Verallgemeinerung dieser Problemstellung und schreiben Sie das entsprechende Programm dafür.
21. Es sei $a, b \in \mathbb{N}$ so dass $a \cdot b = 300$. Wie viele Paare (a, b) gibt es, so dass a und b keinen gemeinsamen Teiler haben? Verallgemeinern Sie die Problemstellung und entwickeln Sie das entsprechende Programm.
22. Jede Ziffer entspricht einem bestimmten Buchstaben. Dann ist KANGAROO + 10000*AROO – 10000*KANG gleich: a) AROOAROO, b) AROOKANG, c) KANGKANG, d) KANGAROO, d) KANGOOO. Finden Sie mit einem Programm heraus, welche Ziffern sich hinter den Buchstaben verbergen.
23. Gegeben ist eine natürliche Zahl N mit neun Ziffern. Entfernen Sie mit einem Programm vier Ziffern aus N derart, dass die resultierende Zahl die kleinstmögliche ist.
24. Lösen Sie mit einem Programm die Gleichungen $x + y = xy$ und $5x + 7y^2 = 1600$.
25. Schreiben Sie ein Programm, um alle natürlichen Zahlenpaare (x, y) auszugeben, für die

$$\frac{1}{x} + \frac{1}{y} = \frac{1}{2} \quad \text{gilt}$$

26. Wie lauten die letzten beiden Ziffern der Zahl 2^n ? Wir nehmen an, dass n in den Datentyp `unsigned long` passt.
27. *Dreieck oder Trapez?* Implementieren Sie ein Programm, das für eine gegebene Zahl n ($1 \leq n \leq 1000$) ein gleichschenkliges Dreieck oder ein gleichschenkliges Trapez wie in den folgenden Beispielen mit Sternchen auf den Bildschirm zeichnet. Jede Zeile unterscheidet sich von der vorhergehenden durch je ein Zeichen '*' an ihren Enden. Falls keine Lösung existiert, geben Sie `Keine Loesung aus`. Beispiele:

n=1	n=4	n=5	n=12	n=16
*	* ***	Keine Loesung.	** **** *****	* *** ***** *****

28. *Kontrollziffer.* Die Kontrollziffer einer eingegebenen natürlichen Zahl N berechnen wir wie folgt: Zuerst bilden wir die Quersumme über N und wenn das

Resultat mehr als eine Ziffer beansprucht, bestimmen wir die Quersumme der Quersumme, und das solange, bis das Resultat nur noch aus einer Ziffer besteht. Beispiel: Die Kontrollziffer von 465329 ist 2. $Quersumme(465329)=29$, $Quersumme(29)=11$, $Quersumme(11)=2$. Wenn wir statt einer Zahl ein Datum in der Form *TT.MM.JJJJ* einlesen, definieren wir die Kontrollziffer über die Zahl *TTMMJJJJ*. Schreiben Sie ein Programm, das die Kontrollziffer für ein gegebenes Datum ermittelt. Beispiel:

Tastatur	Bildschirm
Tag = 23 Monat = 12 Jahr = 1989	Kontrollziffer(23.12.1989)=8

29. *Potenzen von k*. Gegeben sind eine natürliche Zahl k ($2 \leq k \leq 9$) und ein Intervall $[a,b]$. Finden Sie alle natürlichen Zahlen aus dem Intervall $[a,b]$, die Potenzen von k sind ($2 \leq a \leq b \leq 9.000.000$). Beispiel:

Tastatur	Bildschirm
k=2 a=15 b=70	16 32 64

$(16 = 2^4, \quad 32 = 2^5, \quad 64 = 2^6)$

30. Es seien zwei ganze Zahlen a und b und eine natürliche Zahl n gegeben. Schreiben Sie ein Programm, das zuerst prüft, ob a und b teilerfremd sind. Wenn die Antwort "ja" lautet, bestimmen Sie das Paar (x,y) mit den natürlichen Zahlen x und y , so dass $n = ax + by$ gilt.



Abb. 2.7: Winterlandschaft in Ried, Österreich

Einführung in C
Praktisches Lern- und Arbeitsbuch für
Programmieranfänger
Logofătu, D.
2016, XIV, 307 S. 27 Abb., Softcover
ISBN: 978-3-658-12921-7