Ingenieur-informatik

Mitschrift der Vorlesung von Prof. Zacherl im SS 2000

Christian Hampp

(keine Gewähr auf Richtigkeit und Vollständigkeit)

Inhaltsverzeichnis

Ingenieurinformatik

1. Programmaufbau

Grundstruktur:

```
int main ()
{...}
```

main: Schlüsselwort für Hauptprogramm
int: Hauptprogramm gibt ganze Zahl an das Betriebssystem zurück
(): Innerhalb dieser Klammern können dem Hauptprogramm Informationen übergeben werden.
{...}: Block, der die Befehle des Programms enthält.

Einführendes Beispiel:

```
#include <iostream.h>                Einbinden der Ein-/Ausgabefunktionen
int main()
{
/*Programm berechnet Umfang und Fläche des Kreises*/          Kommentar
const float pi=3.14159265;                Konstante
float rad, umf, fla;          Variablen
cout<<"Bitte Radius eingeben! ;
cin>>rad;          Aus-/Eingabe
umf=2*pi*rad;
fla=rad*rad*pi;       Berechnung
cout<<"Fläche: "<<fla<<endl;
cout<<"Umfang: "<<umf;          Ausgabe
return 0;          Festlegen des Rückgabewertes
}          Ende
```

Allgemeine Bemerkungen:

- Groß- und Kleinschreibung werden unterschieden (z.B. fla und Fla sind verschiedene Variablen)
- Alle Schlüsselwörter (reservierten Wörter) müssen klein geschrieben werden. (z. B. main, float, cin, cout,...)
- Jeder Befehl (innerhalb der geschweiften Klammern {}) wird mit ";" abgeschlossen.
- Die Formatierung (zusätzliche Leerzeichen) ändert das Programm nicht. An manchen Stellen ist mindestens ein Leerzeichen zur Trennung notwendig. (z.B. const float)

Kommentare:

- beliebige Zeichenfolge, die eine Erläuterung des Programms enthält.
- können an beliebiger Stelle im Programm stehen

- Kennzeichnung: /*...*/ eingeschlossener Kommentar
  //... Kommentar bis Zeilenende
- werden bei der Ausführung des Programms nicht berücksichtigt
- Kommentare können zum Überspringen von Programmteilen benutzt werden

Konstanten:

- verwendet für Werte, die konstant bleiben
- Typ, Name und Wert einer Konstanten werden nach dem Schlüsselwort const angegeben
- Mehrere Konstanten können durch Wiederholung des Schlüsselwortes const definiert werden oder durch Trennung mit Strichpunkt nach **einem** Schlüsselwort const.

Bsp.: const float pi=3.14159265;
const float e=2.7182818;
const int N=10;

oder const float pi=3.14159265; float e=2.7182818; int N=10;
oder const float pi= ... ,e= .... ;int N=10;

Variable:

- Speicherung / Benennung

Name	Speicherplatzadresse	Werte
rad	10123	1.0
umf	10124 ←	6.28319
fla	10125	3.141592

- Namen und die Datentypen der Variablen müssen zu Beginn des Programms definiert werden.
  z. B. float rad, umf, fla;
  Datentyp Namen
- Anfangswerte können bei der Definition gleich angegeben werden
  float rad=3.7, umf=6.73, fla;
- Variable verschiedenen Datentyps werden durch Strichpunkt getrennt.
  Beispiel: int a, b; oder int a; int b;
  float c, d, e=4.5; float e=4.5

Namen (Identifier):

- Name ist eine Folge von Zeichen bestehend aus Buchstaben, Ziffern und "_" (Underscore / Unterstrich)
- Name beginnt stets mit Buchstabe oder Underscore
- nicht erlaubt sind Sonderzeichen wie: ! Ä Ü Ö ä ö ü $ -
- nicht zulässig sind reservierte Wörter als Namen (z. B. main, float, const, return,...)
- Namen können im Prinzip beliebig lang sein, aber mehr als 10 Zeichen sollten nicht verwendet werden

Beispiele für Namen:

H2SO4	ok	Übung1	f
123A	f	Main	ok
V2A-Stahl	f	otto	ok
V2A_Stahl	ok	0tt0	f
uebung_1	f		

2. Einfache Datentypen

einfach = vom System bereitgestellt
(später: zusammengesetzte Datentypen = selbst definierte Datentypen)

2.1 Ganze Zahlen

Name des Datentyps	**Anzahl der Bits**	**Zahlenbereich**
int	16	-32.768,...,0,...,32.767 (-2^{15} ... 0 ... 2^{15}-1)
unsigned int	16	0,...,65.535 (2^{16}-1)
long	32	-2.147.483.648,...,0,...,2.147.483.647 (-2^{31} ... 0 ... 2^{31}-1)
unsigned long	32	0,...,4.294.967.295

Einschub: Dezimalzahlen – Dualzahlen

Dezimalsystem: 3147 = 3·1000 + 1·100 + 4·10 + 7·1
(Tausender, Einer, Hunderter, Zehner)

- Grundzahl 10
- Stellenwerte: 10^n n = 0, 1, 2, 3, ...
- Ziffern: (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)

Dualsystem: 00101011 = 32 + 8 + 2 + 1 = 43
(2^5 2^3 2^1 / 2^4 2^2 2^0)

- Grundzahl 2
- Stellenwerte: 2^n n = 0, 1, 2, 3, ...
- Ziffern: (0, 1)

Umrechnung Dezimal → Dual
z.B. 115 = 64 + 51 = 64 + 32 +19 = 64 + 32 + 16 +2 + 1
⇒ 1110011

Darstellung ganzer Zahlen:
Anzahl der Bits (Stellen) sei n = 3

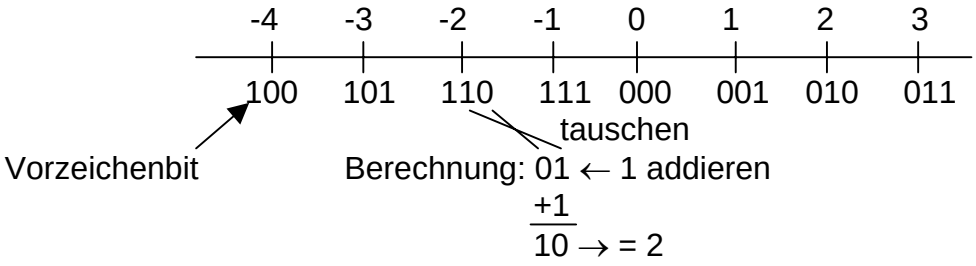
a) positive ganze Zahlen (unsigned ...)

000 = 0	100 = 4	
001 = 1	101 = 5	allgemeiner Zahlenbereich: $0,...,2^n-1$
010 = 2	110 = 6	
011 = 3	111 = 7	

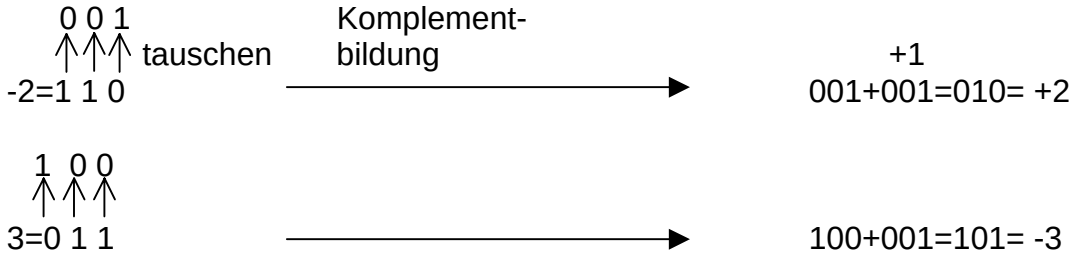
b) positive und negative ganze Zahlen (int, long)

- Kennzeichen des Vorzeichens erforderlich
  $\rightarrow$ benutze z.B. das erste Bit zur Kennzeichnung des Vorzeichens
  0 = + 1 = -
  $\Rightarrow$ 0 hat zwei verschiedene Darstellungen, nämlich 000 ("+0") und 100 ("-0")

Ausweg: Zweierkomplementdarstellung



- allgemeiner Zahlenbereich: $-2^{n-1},...,0,...,2^{n-1}-1$
- Komplementbildung:



Übungsaufgabe:
Zur Darstellung ganzer Zahlen negativ und positiv in Zweierkomplement-darstellung stehen 5 bit zur Verfügung.

a) Wie lautet die Darstellung der Null und der Zahl –1?
00000 = 0 -1 = 11111

b) Welcher Zahlenbereich lässt sich darstellen?
$-2^4,...,0,...,2^4-1 = -16,...,0,...,15$

c) Wie lautet das Bitmuster der kleinsten Zahl?
10000 = -16

2.2 Reelle Zahlen (Gleitkommazahlen)

Name des Datentyps	**Anzahl der Bits**	**Zahlenbereich**	**Genauigkeits-stellen**
float	32	$\pm 3{,}4 \cdot 10^{-38}$ ... $\pm 3{,}4 \cdot 10^{38}$	7
double	64	$\pm 1{,}7 \cdot 10^{-308}$ ... $\pm 1{,}7 \cdot 10^{308}$	15
long double	80	$\pm 3{,}4 \cdot 10^{-4932}$ ... $\pm 3{,}4 \cdot 10^{4932}$	19

Darstellung: ±147.2617e-2 = 1,472617

Vorkommastellen Exponent=10^{-2}

2.3 Zeichen

- Bezeichnung durch das reservierte Wort "char" (character)
- dient zur Darstellung einzelner Zeichen
  - Buchstaben 'a'; 'A'; 'T'
  - Ziffern '0'; '1'; '2'; ...
  - Sonderzeichen '!'; 'ü'; 'ö'; '$'
- Zeichen immer in Hochkommas eingeben

Beispiel:

const char stern = '∗'

Konstanten-schlüsselwort Datentyp Name der Konstanten Wert

char a

Datentyp Variable

a = stern → a = '∗'
a = 'x' → a = 'x'
a = 'a' → a = 'a'

ASCII – Code:

- Frage/ Problem: Wie werden die Zeichen in Bitmuster mit den dualen Ziffern 0 und 1 umgesetzt?
- ASCII – Code gibt die Standardisierung an
- ASCII – Code verwendet zur Darstellung eines Zeichens immer 8 Bit (= 1 Byte)
- Maximal können 256 Zeichen (2^8) dargestellt werden
- Bitmuster 'a': dezimal 97 = 64+32+1
  97 = 01100001
  '8': dezimal 56 = 32+16+8
  = 00111000 = 38hex
  'W': hexadezimal 57 = 01010111 = 87dez
  0101 0111

- nur Teilbereiche (Buchstaben, Ziffern) sind standardisiert

2.4 Logischer Datentyp

- Wird mit dem reservierten Wort "bool" bezeichnet
- kann nur die logischen Werte 0 = false und 1 = true annehmen

3. Operatoren und Standardfunktionen

3.1 Arithmetische Operatoren

Grundrechenarten: Addition: +
Subtraktion: -
Multiplikation: ∗
Division: /

- Verknüpfung von ganzen Zahlen liefert wieder ganze Zahl
  z.B. int / int = int d.h. 3 / 2 = 1
- Verknüpfung von reellen Zahlen liefert immer reelle Zahl
  z.B. 3.0 / 2.0 = 1.5
- Enthält ein Term sowohl ganze als auch reelle Zahlen, so ist das Ergebnis eine reelle Zahl
  z.B. 3.0 / 2 = 1.5

Modulofunktion

- mit dem Zeichen "%" gekennzeichnet

Beispiele:
9 % 5 = 4 (-9) % (-5) = -4
9 % (-5) = 4 23 % 3 = 2
(-9) % 5 = -4

- **nur "int"**
- Die Modulofunktion gibt den ganzzahligen Rest bei der Division ganzer Zahlen an, wobei das Ergebnis das Vorzeichen des Dividenden hat
  z.B. 9 / 5 = 1 Rest 4

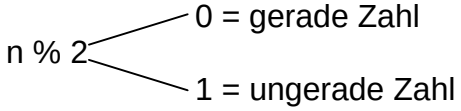
weitere Beispiele:
3 % 10 = 3 0 % 3 = 0
31 % 1 = 0 1 % 3 = 1
9 % 2.0 = (nicht zulässig, da nur für ganze Zahlen definiert
reelle Zahl!

allgemein gilt:

$n \% m = \pm \{0; 1; ...; m-1\}$

↑ Vorzeichen von n

- Zur Unterscheidung von geraden und ungeraden Zahlen verwendet



Regeln für die Auswertung von Termen

- Die Operatoren (/; ∗; %) werden vor den Operatoren (+; -) ausgeführt
- Mehrere Operatoren gleicher Stufe (z.B. /; ∗; %) werden in der Reihenfolge von links nach rechts ausgeführt
- Klammerausdrücke (nur runde Klammern) werden von innen nach außen aufgelöst

Beispiele:
5 + 25 % 6 = 6
(5 + 25) % 6 = 30 % 6 = 0
6 / 2 ∗ 6.0 = 18
6 / (2 ∗ 6.0) = 6 / 12.0 = 0.5
6 / 2 ∗ 6 = 18
6 / (2 ∗ 6) = 6 / 12 = 0 (int-Typ!)

3.2 Mathematische Funktionen

- sind in der Bibliothek <math.h> enthalten, die zu Programmbeginn eingebunden werden muss

#include <math.h>

Wichtigste Funktionen:

- Potenzfunktion: x^y = pow(x, y)
- Quadratwurzel: $\sqrt{x}$ = sqrt(x)
- Sinus: sin (x)
- Cosinus: cos(x)

} im Bogenmaß

- Exponentialfunktion: exp(x)
- natürlicher Logarithmus: log(x)
- Betrag: fabs(x)

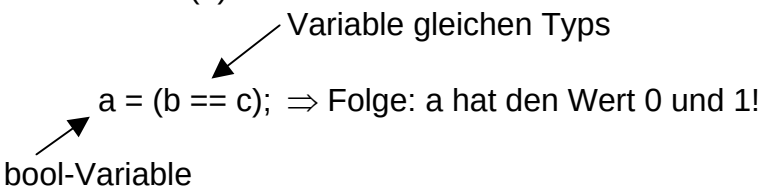
3.3 Logische Operatoren

a) Vergleichsoperatoren

Operator	Bedeutung
==	gleich
<	kleiner
<=	kleiner-gleich
>	größer
>=	größer-gleich
!=	ungleich

- werden auf Variablen/ Konstanten gleichen Typs angewendet, wobei auch ganze Zahlen mit reellen Zahlen verglichen werden können
- Ergebnis ist immer vom Datentyp "bool"; d.h. true (1) oder false (0)

Beispiele: 3.14 > 3 true (1) 3.14 != 3 true (1)
'A' < 'B' true (1) (Reihenfolge nach ASCII-Code)
5 == 4 false (0)



b) Verknüpfungsoperatoren

- verknüpfen logische Aussagen/ Werte
- Operatoren: && logisches Und (And)
  || logisches Oder (Or)
  ! Negation

Wahrheitstabellen:

&&	f	t
f	f	f
t	f	t

||	f	t
f	f	t
t	t	t

!	f	t
	t	f

Beispiel: (a > 0) && (a <= 2) **nicht:** 0 < a <= 2

4. Verzweigungen

4.1 If-Anweisung

Beispiel: Berechnung der Quadratwurzel

```
#include <math.h>
#include <iostream.h>
int main()
{
/* Programm berechnet Quadratwurzel einer Zahl*/
/* Bei negativer Zahl wird Fehlermeldung ausgegeben*/
float zahl,ergeb;
cout<<"Bitte Zahl eingeben!";
cin>>zahl;
if (zahl<0)
{
cout<<"Quadratwurzel kann nicht berechnet werden.";
cout<<"Zahl ist negativ!";
}
else
{
ergeb = sqrt(zahl);
cout<<"Ergebnis:"<<ergeb;
}
return 0;
}
```

Allgemeines Schema:

```
If (Bedingung)
{
Anweisungsblock 1
}
else
{
Anweisungsblock 2
}
```

Bemerkungen:

- Bedingung darf nur die Werte false (0) und true (1) annehmen
- true: **nur** Anweisungsblock 1 wird ausgeführt
- false: **nur** Anweisungsblock 2 wird ausgeführt
- else-Zweig kann entfallen z.B. bei Zahl >= 0 erfolgt überhaupt keine Aktion

Weiteres Beispiel: Minimum von 3 reellen Zahlen

Schreiben Sie ein Programm, das 3 reelle Zahlen einliest, das Minimum berechnet und ausgibt.

```
#include <iostream.h>
int main()
{
float a,b,c;
cout<<"Bitte 3 reelle Zahlen eingeben!"<<endl;
cin>>a>>b>>c;
if ((a<=b) && (a<=c))
{
cout<<"Minimum ist:"<<a;
}
if ((b<=a) && (b<=c))
{
cout<<"Minimum ist:"<<b;
}
if ((c<=a) && (c<=b))
{
cout<<"Minimum ist:"<<c;
}
return 0;
}
```

alternative Lösung:

```
float min;

min = a;
if (b<=a);
{
min = b;
}
if (c<=min);
{
min = c;
}
```

Geschachtelte If-Anweisung

```
if (Bedingung 1)
{
    if (Bedingung 2)
    {
    Anweisung 1
    }
    else
    {
    Anweisung 2
    }
}
else
{
if (Bedingung 3)
    {
    Anweisung 3
    }
    else
    {
    Anweisung 4
    }
}
```

Bedingung 1	Bedingung 2	Bedingung 3	Anweisung
t	t	t	1
t	t	f	1
t	f	t	2
t	f	f	2
f	t	t	3
f	t	f	4
f	f	t	3
f	f	f	4

Weitere Beispiele: Schaltjahr
Schreiben Sie ein Programm, das eine Jahreszahl einliest, und angibt, ob es sich um ein Schaltjahr handelt, oder nicht.

1600	1700	1800	1900	1904	...	1996	2000	2004	2100
j	n	n	n	j		j	j	j	n

Regel: Es liegt ein Schaltjahr vor, wenn die Jahreszahl durch 4, aber nicht durch 100 teilbar ist, oder wenn die Jahreszahl durch 400 teilbar ist.

```
#include <iostream.h>
int main()
{
int jahr;
cout<<"Geben Sie die Jahreszahl ein!";
cin>>jahr;
if ((jahr%4==0) && (jahr%100!=0))
{
cout<<"Schaltjahr";
}
else
{
if (jahr%400==0)
{
cout<<"Schaltjahr";
}
else
{
cout<<"Kein Schaltjahr";
}
}
return 0;
}
```

weitere Lösung:

```
if((jahr%4==0) && (jahr%100!=0)) || (jahr%400==0)
{
cout<<"Schaltjahr";
}
else
{
cout<<"Kein Schaltjahr";
}
```

4.2 Switch-Anweisung (Mehrfachanweisung)

Beispiel: Schreiben Sie ein Programm, das die Punktwertung der Kollegstufe in Zeugnisnoten umsetzt.

Punkte	Note
0	ungenügend
1, 2, 3	mangelhaft
4, 5, 6	ausreichend
7, 8, 9	befriedigend
10, 11, 12	gut
13, 14, 15	sehr gut

a) Lösung mit if-Anweisung:

```
    |
{
int pun
cout<<"Geben Sie die Punktezahl ein!";
cin>>pun;
if(pun==0)
{
cout<<"ungenügend";
}
    |
if((pun==1) || (pun==2) || (pun==3))
{
cout<<"mangelhaft";
}
    |
    |
```

b) Lösung mit Switch-Anweisung:

```
#include<iostream.h>
int main()
{
int pun;
cout<<"Geben Sie die Punktezahl ein!";
cin>>pun;
switch (pun)
{
case 0: cout<<"ungenügend"; break;
case 1: case 2: case 3: cout<<"mangelhaft"; break;
case 4: case 5: case 6: cout<<"ausreichend"; break;
    |
    |
case 13: case 14: case 15: cout<<"sehr gut"; break;
```

```
default: cout<<"Unzulässige Eingabe"; break;
}
return 0;
}
```

Allgemeines Schema

```
switch(Ausdruck)
{
case wert11: case wert12: ... Anweisung 1; break;
case wert21: case wert22: ... Anweisung 2; break;


case wert n1: case wert n2: ... Anweisung n; break;
default: alternative Anweisung; break;◄— (kann weggelassen werden)
```

Bemerkungen:

- Ausdruck ist vom Datentyp Integer oder Character (nicht zulässig: float, double, d.h. reelle Zahlen)
- Werte müssen ausgerechnet sein oder als Konstante definiert sein (keine Variablen)
- Default-Zweig kann weggelassen werden. Trifft dann kein Wert zu, wird keine Anweisung ausgeführt.
- Break-Anweisung führt zum Ende der Verzweigung

Weiteres Beispiel: Römische Ziffern

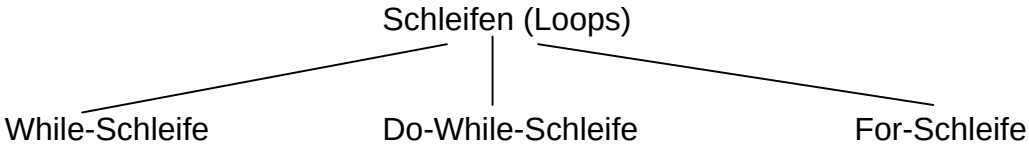
Schreiben Sie ein Programm, das eine römische Ziffer einliest und den zugehörigen Dezimalwert ausgibt.

Es gilt:	I = 1	C = 100
	V = 5	D = 500
	X = 10	M = 1000
	L = 50	

```
#include <iostream.h>
int main()
{
char rom;
int dez;
cout<<"Geben Sie eine römische Zahl ein!";
cin>>rom;
dez = 0;
switch(rom)
{
case 'I': dez = 1; break;
case 'V': dez = 5; break;
case 'X': dez = 10; break;
case 'L': dez = 50; break;
case 'C': dez = 100; break;
```

```
case 'D': dez = 500; break;
case 'M': dez = 1000; break;
default: cout<<"Unzulässige Eingabe";
}
if (dez>0)
cout<<"Die Dezimalzahl heißt:"<<dez;
return 0;
}
```

5. Schleifen (Loops)



5.1 While-Schleife

Schreiben Sie ein Programm, das die Quadrate der Zahlen 1 bis 20 auf den Bildschirm schreibt!
also 1 4 9 16 25 ...

```
#include <iostream.h>
#include <math.h>
int main()
{
int zahl, qua;
zahl = 1;
while(zahl<=20)
{
qua = pow(zahl,2);
cout<<qua<<"  ";
zahl = zahl + 1;
}
return 0;
}
```

Allgemeines Schema:

```
while(Bedingung)
{
Anweisungen der Schleife
}
```

Bemerkungen:

- Anweisungen werden solange ausgeführt, bis die Bedingung falsch wird
- Ist die Bedingung beim ersten Anlauf falsch, wird die Schleife ganz übersprungen

5.2 Do-While-Schleife

Allgemeines Schema:

```
do
{
Anweisungen der Schleife
}
while(Bedingung);
```

Unterschiede zur While-Schleife:

- Bedingung wird am Ende der Schleife geprüft
  → Schleife wird mindestens einmal durchlaufen
- oft benutzt - um zulässige Dateneingabe sicherzustellen
  - um Programme ohne Neustart zu wiederholen

Beispiel: Wurzelberechnung

```
#include <iostream.h>
#include <math.h>
int main()
{
float zahl, ergeb;
char antwort;
do
{
do
{
cout<<"Bitte Zahl >= 0 eingeben";
cin>>zahl;
}
while (zahl<0);
ergeb=sqrt(zahl);
cout<<"Wurzel aus "<<zahl<<" ist: "<<ergeb<<endl;
cout<<"Weitermachen (j/n) ?"<<endl;
cin>>antwort;
}
while (antwort=='j');
return 0;
}
```

5.3 For-Schleife

Beispiel: Berechnung der Fakultät

$n \in \mathbb{N}$: $n! = 1 \cdot 2 \cdot 3 \cdot ... \cdot n$

z.B. $4! = 24$

$1! = 1$

```
/* Fakultätsberechnung */
#include <iostream.h>
int main()
{
int n, i;
long fakul;
fakul=1;
cout<<"Bitte eine ganze Zahl eingeben!";
cin>>n;
for(i=2; i<=n; i=i+1)
{
fakul=fakul*i;
}
cout<<"Fakultät von "<<n<<" ist: "<<fakul<<endl;
return 0;
}
```

allgemeines Schema:

```
for(Anfangswert; Bedingung; Veränderung)
{
Anweisungen der Schleife
}
```

Bemerkungen:

- Für Anfangswert, Bedingung und Veränderung sind ganze oder reelle Zahlen erlaubt.
  z.B. float i
  ```
  for (i=1.0; i<13; i=i+2.5)
  {
  ...
  }
  ```
- Der Datentyp der Laufvariablen (bisher i) kann auch erst **in** der Schleife definiert werden.
  z.B. for (float i=1.0; i<13; i=i+2.5)
- Wird die Laufvariable um 1 erhöht bzw. erniedrigt, so können folgende Abkürzungen benutzt werden:
  i=i+1 → i++
  i=i-1 → i- -
- Erfüllt der Anfangswert die Bedingung nicht, so wird die Schleife ganz übersprungen.
  z.B. for (int i=3; i<=2; i- -)

Geschachtelte For-Schleifen

```
z.B. for (int i=1; i<=2; i++)
       {
       for (int j=1; j<=3; j++)
         {
```

```
    for (int k=4; k>=1; k- -)
      {
      cout
      }
    }
  }
```

Bemerkung: Geschachtelte Schleifen werden von innen nach außen abgearbeitet

Ausgabe:
```
114
113
112
111
124
123
122
121
134
133
132
131
214
213
212
211
...
...
```

Übungsaufgabe: Welche Ausgabe liefert das folgende Programm?

```
#include <iostream.h>
int main()
{
int kount=0, summe=0;
for (int i=1; i<=3; i++)
{
for (int j=i; j<=3; j++)
{
for (int k=j-i; k<=j; k++)
{
cout<<i<<j<<k<<endl;
kount=kount+1;
summe=summe+k;
}
}
}
cout<<kount<<endl;
cout<<summe<<endl;
return 0;
}
```

i	j	k	Ausgabe
1	1	0, 1	110
			111
	2	1, 2	121
			122
	3	2, 3	132
			133
2	2	0, 1, 2	220
			221
			222
	3	1, 2, 3	231
			232
			233
3	3	0, 1, 2, 3	330
			331
			332
			333

kount = 16 = Anzahl der Ausgaben
summe = 24 = Summe der k-Werte

Weitere Beispiele zu Schleifen:

1) Größter gemeinsamer Teiler nach Euklid:
   Gegeben seien: a, b $\in$ *IN*
   gesucht ggT(a, b)

   a = 36 = 2 · 18 = 2 · 2 · 9 = **2** · **2** · **3** · 3
   b = 84 = 2 · 42= 2 · 2 · 21= **2** · **2** · **3** · 7
   } ggT(36, 84) = 12

Euklidischer Algorithmus

1) ggT(a, a) = a
2) Der größte gemeinsame Teiler bleibt unverändert, wenn die größere der beiden Zahlen durch die Differenz der beiden Zahlen ersetzt wird.

   also: ggT(a, b) = ggT(a, b – a) für b > a
   ggT(a, b) = ggT(a – b, b) für b < a
   ggT(36,84)= ggT(36,48)= ggT(36,12)= ggT(24,12)= ggT(12,12)= **12**

Programm:

```
/*Euklidischer Algorithmus*/
#include <iostream.h>
int main()
{
int a, b, am, bm;
cout<<"Bitte 2 natürliche Zahlen eingeben!";
cin>>a>>b;
```

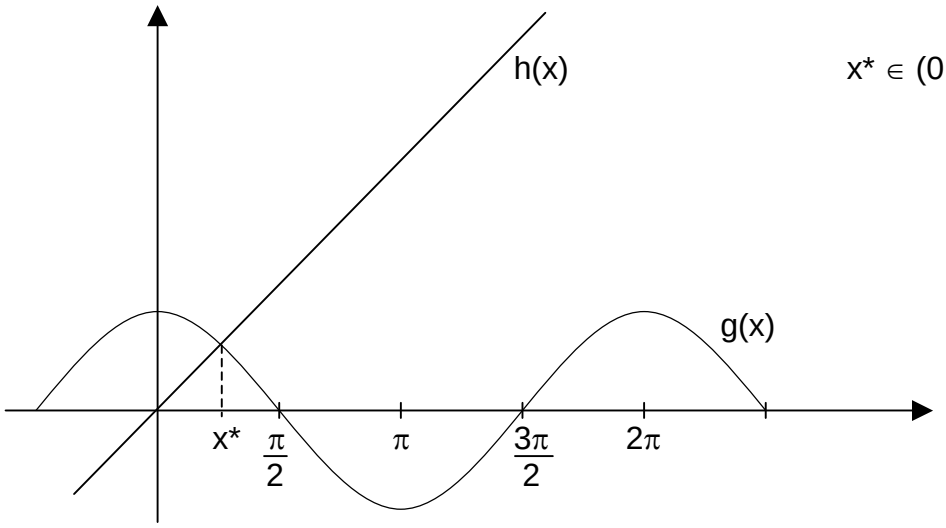
```
am=a; bm=b;
while(a!=b)
{
if(b>a)
{
b=b-a;
}
else
{
a=a-b;
}
}
cout<<"GGT von "<<am<<" und "<<bm<<" ist: "<<b;
return 0;
}
```

Nullstellenbestimmunng nach dem Bisektionsverfahren

Gesucht: Nullstelle der Funktion: f(x) = 2x – cos x
Frage: Hat die Funktion überhaupt eine Nullstelle?

$$f(x) = \underbrace{2x}_{h(x)} - \underbrace{\cos x}_{g(x)}$$

Geometrische Betrachtung:



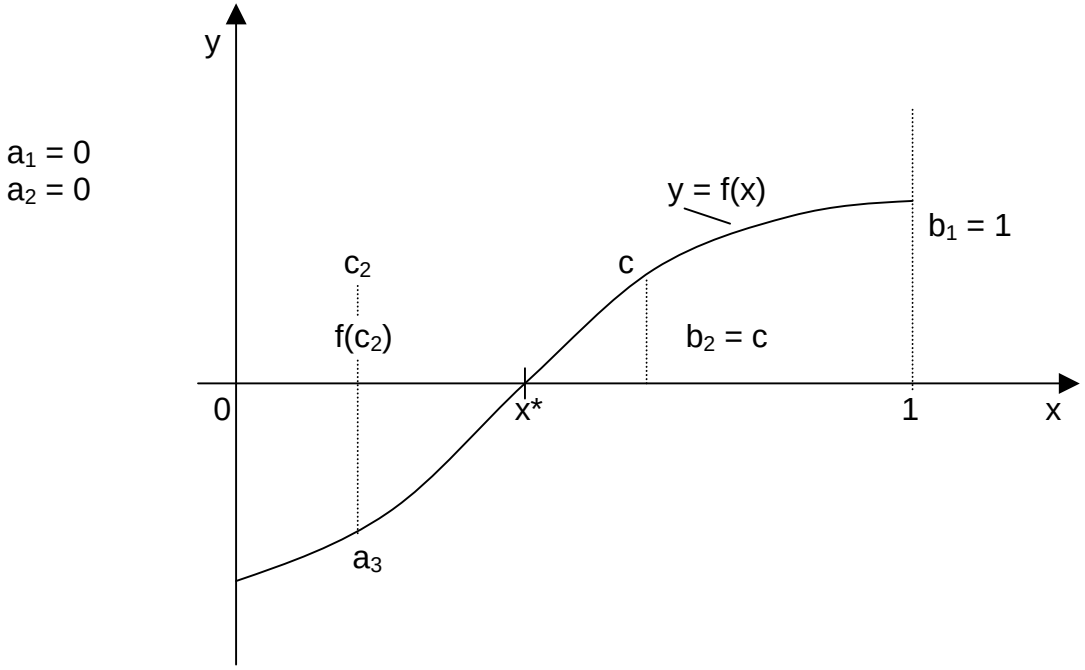
$x^* \in (0, 1)$

$f(0) = 2 \cdot 0 - \cos 0 = -1 < 0$

$f(1) = 2 \cdot 1 - \cos 1 = 1{,}46 > 0$

} f hat im Intervall mind. eine Nullstelle x*

Bisektionsverfahren:



Algorithmus:

1) Wähle Intervallgrenzen a, b

2) Halbiere das Intervall $c = \frac{a+b}{2}$

3) Berechne f(c)

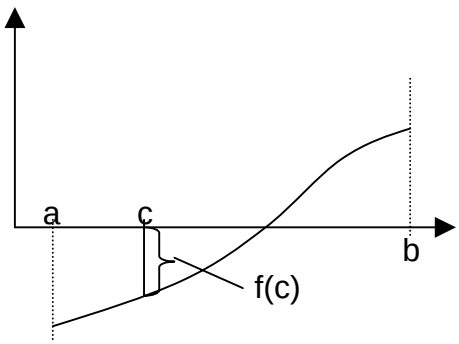
4) Wenn f(c) > 0, dann b = c
   Wenn f(c) < 0, dann a = c

5) Wiederholung des Verfahrens ab 2) bis $|a - b| < 10^{-5}$ oder $f(c) < 10^{-5}$

Nr.	a	b	c	f(c)
1	0	1	0,5	0,12 > 0
2	0	0,5	0,25	-0,47 < 0
3	0,25	0,5	0,375	-0,18 < 0
4	0,375	0,5	0,4375	-0,03 < 0
...	...	...	...	...

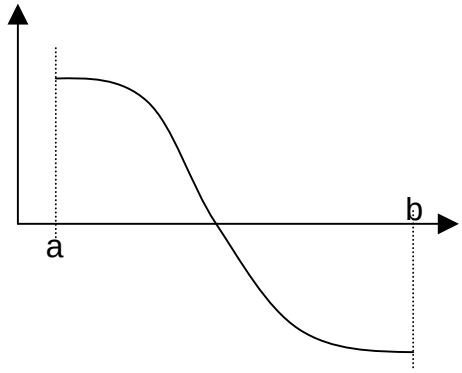
Allgemeine Bedingung für das Bisektionsverfahren

1. Fall: f(a) < 0 und f(b) > 0



Bedingung: Wenn f(c) > 0, dann b = c, sonst a = c

2. Fall: f(a) > 0 und f(b) < 0



Bedingung: Wenn f(c) < 0, dann b = c, sonst a = c

Allgemeine Bedingung:

Wenn f(b) · f(a) > 0, dann b = c, sonst a = c, d.h. Funktionswerte haben gleiches Vorzeichen;

oder: Wenn f(a) · f(c) < 0, dann b = c, sonst a = c, d.h. Funktionswerte haben verschiedene Vorzeichen

5.4 Kontrollbefehle "break" und "continue"

- Die Kontrollbefehle "break" und "continue" können bei allen 3 Schleifentypen angewandt werden.
- "break": Schleife (als Ganzes) wird beendet.
- "continue": Rest der Anweisungen der Schleife werden übersprungen und die Schleife wird mit dem nächsten Durchgang weitergeführt.

Beispiel: Auswahl unter verschiedenen Programmen (Menüerstellung!)

```
#include <iostream.h>
int main()
{
char zeich;              =true (Schleife läuft immer)
while(1)
  {
  cout<<"Wählen Sie: a, b; x = Ende:";
  cin>>zeich;
  if (zeich=='a')
    {
    cout<<"Programm a"<<endl;
    continue;
    }
  if (zeich=='b')
    {
    cout<<"Programm b"<<endl;
    continue;
    }
```

```
  if (zeich=='x')
    {
    break;
    }
cout<<"Falsche Eingabe!"<<endl;
return 0;
}
}
```

6. Arrays (Felder)

• bisher nur Grunddatentypen

- int (long)
- float (double; long double)
- char
- (bool)

• jetzt: selbstdefinierte Datentypen

Beispiele: Vektor: $\vec{a} = (1.0; 2.0; -1.5)$ → float a[3]

Matrix: $A = \begin{pmatrix} 1 & 0 & 2 & 3 \\ 4 & 2 & 0 & 1 \\ 1 & -1 & 2 & 4 \end{pmatrix}$ → int A[3][4]

Spaltenzahl

Zeilenzahl

Arraydefinition: Grundtyp (int; float; char) — Arrayname (Variablenname) — [dim 1] [dim 2]...[dim n] (Dimensionen, Indexbereiche)

Bemerkungen:

- Die Dimensionen dim 1; dim 2; ...; dim n müssen natürliche Zahlen sein und mit festen Werten belegt sein (variable Dimensionierung bei der Definition nicht möglich)
- Zählweise der Indexbereiche: float a[3] → a = (a_0, a_1, a_2)

$$\text{int A[3][4]} \rightarrow A = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \end{bmatrix}$$

- Indexzählung beginnt immer bei 0!

- Wertzuweisung der Arrays:

  1. Möglichkeit: direkt bei der Definition der Variablen

     z.B. float a[3] = {1.0, -2.5, 3.14};

     int b[2][3] = {{4, 2, 1}, {3, -1, 0}};

$$\left(b = \begin{pmatrix} 4 & 2 & 1 \\ 3 & -1 & 0 \end{pmatrix} \right)$$

2. Möglichkeit: einzelne Wertzuweisungen im Programm

```
float a[3];
    .
    .
a[0] = 1.0;
a[1] = -2.5;
a[2] = 3.14;
```

```
int b[2][3];
    .
    .
b[0][0] = 4;
b[0][1] = 2;
    .
    .
```

Ein- und Ausgabe von Arrays

Regel: Arrays können nur elementweise eingelesen und ausgegeben werden.

Beispiel: Schreiben Sie ein Programm, das einen Vektor der Länge 4 einliest, jedes Vektorelement mit 2 multipliziert und das Ergebnis ausgibt.

$$\vec{a} = (1,4,3,2) \xrightarrow{\cdot 2} \vec{b} = (2,8,6,4)$$

```
#include <iostream.h>
int main()
{
int a[4], b[4];
cout<<"Bitte Vektor eingeben!";
cin>>a[0]>>a[1]>>a[2]>>a[3];          Eingabe: 1 4 3 2
for (int i=0; i<=3; i++)
{
b[i]=2*a[i];
}
cout<<"Ergebnis: "<<b[0]<<" "<<b[1]<< " "<<b[2]<< " "<<b[3];
return 0;
}
```

- Wird auf ein Arrayelement ausserhalb des zulässigen Indexbereiches zugegriffen, so erfolgt weder beim Compilieren noch bei der Programmausführung eine Fehlermeldung.

z.B.: int a[3]; → a[3] → (a_0, a_1, a_2)

a[3], a[4] } beliebige Speicherinhalte

2	4	3	6
a_0	a_1	a_2	a_3

Wenn For(int i=1, i<=4, i++) ⇒ $b_1 = 2a_1$

$b_2 = 2a_2$

$b_3 = 2a_3$

Aufgaben zu den Arrays

1) Multiplikation einer Matrix mit einem Vektor

$$\underbrace{\begin{pmatrix} 1 & 2 & 1 \\ 1 & -1 & 0 \\ 2 & 1 & 1 \end{pmatrix}}_{A} \underbrace{\begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}}_{\vec{x}} = \begin{pmatrix} 1\cdot 1+2\cdot 0+1\cdot 2 \\ 1\cdot 1+(-1)\cdot 0+0\cdot 2 \\ 2\cdot 1+1\cdot 0+1\cdot 2 \end{pmatrix} = \underbrace{\begin{pmatrix} 3 \\ 1 \\ 4 \end{pmatrix}}_{\vec{b}}$$

allgemein:

$$\begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_0 \\ b_1 \\ b_2 \end{pmatrix}$$

$b_0 = a_{00} \cdot x_0 + a_{01} \cdot x_1 + a_{02} \cdot x_2$
$b_1 = a_{10} \cdot x_0 + a_{11} \cdot x_1 + a_{12} \cdot x_2$
$b_2 = a_{20} \cdot x_0 + a_{21} \cdot x_1 + a_{22} \cdot x_2$

allgemein: $b_i = a_{i0} \cdot x_0 + a_{i1} \cdot x_1 + a_{i2} \cdot x_2 = \sum_{j=0}^{2} a_{ij} x_{ij}$

```
#include <iostream.h>
int main()
{
int x[3], b[3];
int a[3][3]={{1,2,1}, {1,-1,0}, {2,1,1}};
cout<<"Bitte Vektor eingeben!";
cin>>x[0]>>x[1]>>x[2];
for (int i=0; i<=2; i++)
{
b[i]=0;
for (int j=0; j<=2; j++)
{
b[i]=b[i]+a[i][j]*x[j];
}
}
cout<<"Ergebnis: "<<b[0]<<" "<<b[1]<<" "<<b[2];
return 0;
}
```

2) Maximum einer Folge von Zahlen

Schreiben Sie ein Programm, das n Zahlen einliest, das Maximum dieser Zahlen bestimmt und ausgibt. Sei n = 5

Idee: Speichere die Folge der 5 Zahlen als Vektor ab.
Bestimme das Maximum mit Hilfe des Vektors.

```
#include <iostream.h>
int main()
{
int a[5], amax;
cout<<"Bitte 5 ganze Zahlen eingeben!";
cin>>a[0] >>a[1] >>a[2] >>a[3] >>a[4];
amax=a[0];
for (int i=1; i<=4; i++)
{
if (a[i]>amax)
{
amax=a[i];
}
}
cout<<"Maximum: "<<amax;
return 0;
}
```

7. Unterprogramme (Funktionen)

Unterprogramm = selbständige Programmeinheit, die eine Teilaufgabe ausführt.

Beispiel: Definieren Sie die Funktion $y = f(x) = 3x^3 + 4x^2 – 7x + 1$ als Unterprogramm.

```
#include <iostream.h>
#include <math.h>
float poly(float x)
{
float erg;
erg = 3*pow(x, 3)+4*pow(x, 2)-7*x+1;
return erg;
}
int main()
{
float x, y;
cout<<"Bitte x eingeben: ";
cin>>x;
y=poly(x);
cout<<"Ergebnis: "<<y;
return 0;
}
```

7.1 Allgemeines Schema für Unterprogramme

```
Rückgabetyp        Funktionsname / (Parameterliste)
                   Unterprogrammname
{
   ⋮
return
}
```

Bemerkungen:

1) Als Rückgabetyp sind die Standarddatentypen möglich:
   - int, long
   - float, double
   - char
   - (bool)

Zusätzlich kann void = neutral, leer als Rückgabetyp verwendet werden. In diesem Fall wird **kein** Wert zurückgegeben, return entfällt.

```
z.B.: void gruss()
      {
      cout<<"Hallo, wie geht's?";
      }
```

Spezialfall: Hauptprogramm

- Hauptprogramm wird vom Betriebssystem auch als Unterprogramm angesehen.

```
→     int main()          oder     void main()
      {                            {
        ⋮                            ⋮
      return 0;                    kein Return!
      }                            }
```

2) Die Parameterliste kann leer sein, einen oder mehrere Parameter enthalten. Auch bei leerer Parameterliste ist die runde Klammer zu setzen.

z.B.: $u(x, y, z) = x^2 + y^2 + z^2$

```
float u (float x, float y, float z)

{                                  Einzelaufzählung mit Typ notwendig!
float erg;
erg=x*x+y*y+z*z;   } return x*x+y*y+z*z
return erg;
}
```

3) Unterprogramme können auch nach dem Hauptprogramm definiert werden.

```
float poly(float x);   <-- Deklaration des Unterprogramms
                           wichtig
int main()
{                      } Hauptprogramm
return 0;
}
```

```
float poly(float x)   <-- Definition des Unterprogramms
{
float erg;
erg=3*pow(x, 3)+4*pow(x, 2)-7*x+1;
return erg;
}
```

Weitere Beispiele:

1) Abschnittsweise definierte Funktionen

$$\text{Sei} \quad f(x) = \begin{cases} e^x & \text{für } x < 0 \\ \cos(\pi/2\, x) & \text{für } 0 \le x < 1 \\ \ln x & \text{für } x \ge 1 \end{cases}$$

```
#include <iostream.h>
#include <math.h>
float abschnitt(float x)
{
float ergeb;
const float pi=3.1415927;
if (x<0)
{
ergeb=exp(x);
}
if ((x>=0)&&(x<=1))
{
ergeb=cos(pi/2*x);
}
if (x>=1)
{
ergeb=log(x);
}
return ergeb;
}
void main()
{
float x, y;
cout<<"Bitte x eingeben! ";
cin>>x;
y=abschnitt(x);
```

```
cout<<"Ergebnis: "<<y;
}
```

2) Fakultät einer natürlichen Zahl

$n \in \mathbb{N}$

$n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot ... \cdot n$

$0! = 1$

```
#include <iostream.h>
long fakul(int n);
void main()
{
int n;
long y;
do
  {
  cout<<"Bitte natürliche Zahl >= 0 eingeben!";
  cin>>n;
  }
  while (n<0);
y=fakul(n);
cout<<"Ergebnis: "<<y;
}
long fakul(int n)
{
long ergeb;
if(n==0)
  {
  ergeb=1;
  }
else
  {
  ergeb=1;
  for(int i=1; i<=n; i++)
    {
    ergeb=ergeb*i;
    }
  }
return ergeb;
}
```

Nachtrag zur Kreiszahl π: π ist als Konstante unter dem Namen M_PI in der Datei <math.h> verfügbar.

2.7 Werte und Variablenparameter

void f(int x, int y, float z, int & u, int & v, double & w)

Eingabeparameter — Rückgabeparameter

Eingabeparameter: (Werteparameter)
- ohne das Zeichen "&"
- vor dem Aufruf mit Werten zu besetzen
- **call by value**
- Wert wird auf die lokale Variable im Unterprogramm kopiert
- keine Rückwirkung auf die Variablen im aufrufenden Hauptprogramm

Rückgabeparameter: (Variablenparameter)
- mit dem Zeichen "&"
- beim Aufruf mit Variablen zu besetzen
- **call by reference**
- lokale Variable im Unterprogramm und Variable im aufrufenden Programm werden verknüpft
- Rückwirkung auf Variable im Hauptprogramm
- Rückgabe von mehreren Ergebniswerten

Beispiele:

1) Zeitumrechnung

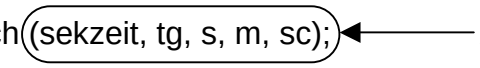
Schreiben Sie ein Unterprogramm, das eine Zeiteingabe in Sekunden einliest und in Tage, Stunden, Minuten und Sekunden umwandelt.

z.B.: 124349 s / 60 = 2072 min **29s**
2072 min / 60 = 34 h **32 min**
34 h / 24 = **1 d** **10 h**

```
/*Zeitumrechnung*/
#include <iostream.h>
#include <math.h>
void umrech(long zeit, long & tag, long & std, long & min, long & sec)
{
long rest;
sec=zeit%60;
rest=zeit/60;
min=rest%60;
rest=rest/60;
std=rest%24;
tag=rest/24;
}
int main()
{
long sekzeit;
long tg, s, m, sc;
```

```
cout<<"Geben Sie die Zeit in Sekunden an!";
cin>>sekzeit;

umrech(sekzeit, tg, s, m, sc);
```



Reihenfolge wichtig!

```
cout<<"Tage:"<<tg<<endl;
cout<<"Stunden:"<<s<<endl;
cout<<"Minuten:"<<m<<endl;
cout<<"Sekunden:"<<sc<<endl;
return 0;
}
```

2) Vertauschen zweier Zahlen

Schreiben Sie ein Unterprogramm, das 2 reelle Zahlen vertauscht!

```
/*Zahlentausch*/
#include <iostream.h>
void tausch(float & x, float & y)
{
float ablage;
ablage=x;
x=y;
y=ablage;
}
void main()
{
float a, b;
cout<<"Geben Sie zwei Zahlen ein!";
cin>>a>>b;
tausch(a, b);
cout<<"Ergebnis: "<<a<<" "<<b;
}
```

Tauschen ohne Ablage:

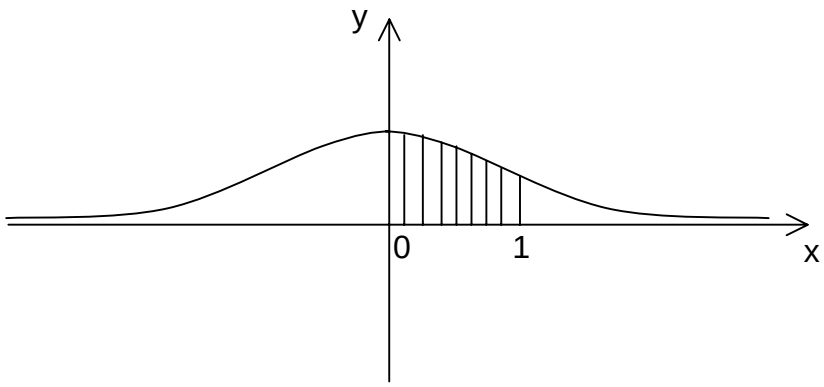
```
void tausch(float & x, float & y)
{
x=x+y;
y=x-y;
x=x-y;
}
```

oder:

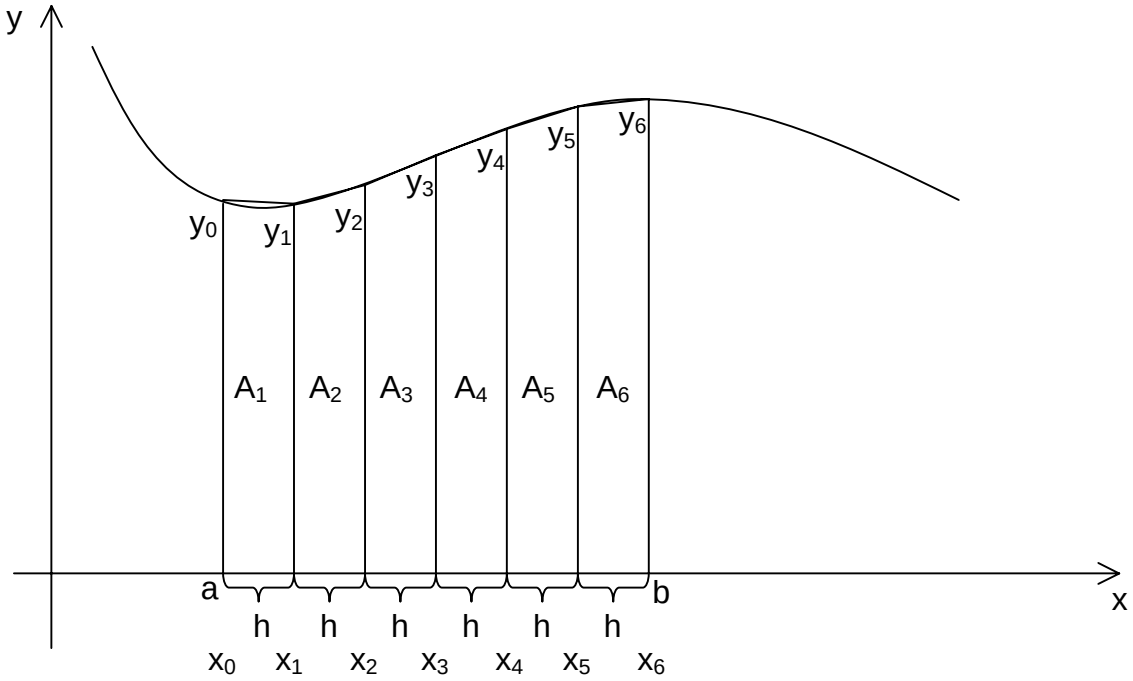
```
void tausch(float & x, float & y)
{
x=x*y;
y=x/y;
x=x/y;
}
```

3) Integration (Trapezregel)

Aufgabe: Berechnen Sie das bestimmte Integral $\int_a^b f(x)dx$, z.B. $\int_0^1 e^{-x^2} dx$ ist nur numerisch lösbar.



Trapezformel: Sei $f(x) \geq 0$ für alle $x \in \mathbb{R}$



Zerlege das Integrationsintervall [a, b] in n Teilintervalle mit der Länge h:

$$h = \frac{b-a}{n}$$

Stützstellen: $x_i = a + i \cdot h$ (i = 0, 1, 2, 3, ..., n)

Funktionswert: $y_i = f(x_i) = f(a + i \cdot h)$

speziell: $y_0 = f(a)$ $y_n = f(b)$

Teilflächen / Trapez:

$$A_1 = h \cdot \frac{y_0 + y_1}{2}$$

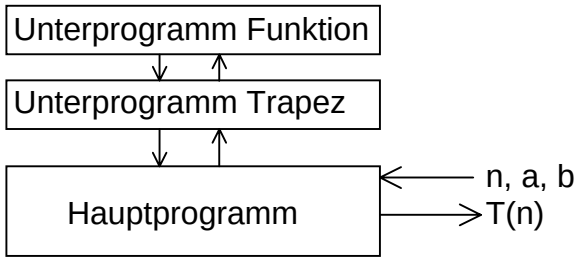
$$A_2 = h \cdot \frac{y_1 + y_2}{2}$$

$$A_{n-1} = h \cdot \frac{y_{n-2} + y_{n-1}}{2}$$

$$A_n = h \cdot \frac{y_{n-1} + y_n}{2}$$

damit $\int_a^b f(x)dx \approx \frac{h}{2}\left(y_0 + y_n + 2 \cdot \sum_{i=1}^{n-1} y_i \right) = \frac{h}{2}\left(f(a) + f(b) + 2 \cdot \sum_{i=1}^{n-1} f(a + i \cdot h) \right)$

Programmstruktur:



Zur Erinnerung: Werte- und Variablenparameter

void f (int x, float y, float & z, float & u)

Eingabeparameter	Ausgabeparameter
- Werte	- Variable
- call by value	- call by reference

Spezialfall: Arrays als Parameter

- **Arrays werden immer automatisch als Variablenparameter behandelt (kein "&" – Zeichen setzen)**

Beispiel: Schreiben Sie ein Unterprogramm, das die Determinante einer 2×2 Matrix berechnet und die transponierte Matrix zurückgibt.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \rightarrow \det A = a_{11} \cdot a_{22} - a_{12} \cdot a_{21} \quad (= a_{00} \cdot a_{11} - a_{01} \cdot a_{10})$$

$$A^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \end{bmatrix}$$

hier kein "&", obwohl Rückgabeparameter

```
#include <iostream.h>
void dettrans(int a[2][2], int & det, int at[2][2])
{
det=a[0][0]*a[1][1]-a[1][0]*a[0][1];
at[0][0]=a[0][0];
at[1][0]=a[0][1];
at[0][1]=a[1][0];
at[1][1]=a[1][1];
}

int main()
{
int a[2][2];
int d;
int b[2][2];
for (int i=0; i<=1; i++)
{
cout<<"Geben Sie die "<<i+1<<"-te Zeile ein!";
cin>>a[i][0]>>a[i][1];
}
dettrans(a, d, b);
cout<<"Determinante der Matrix: "<<d<<endl;
cout<<"Transponierte Matrix: "<<endl;
for (i=0; i<=1; i++)
{
cout<<b[i][0]<<" "<<b[i][1]<<endl;
}
return 0;
}
```

7.2 Rekursion

bisher: - Hauptprogramm ruft Unterprogramm auf
- Ein Unterprogramm ruft ein anderes Unterprogramm auf

Rekursion: Ruft ein Unterprogramm sich selbst auf, so spricht man von **Rekursion.**

Beispiel: Berechnen der Fakultät

$n \in \mathbb{N}: n! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot ... \cdot n$

speziell $0! = 1$

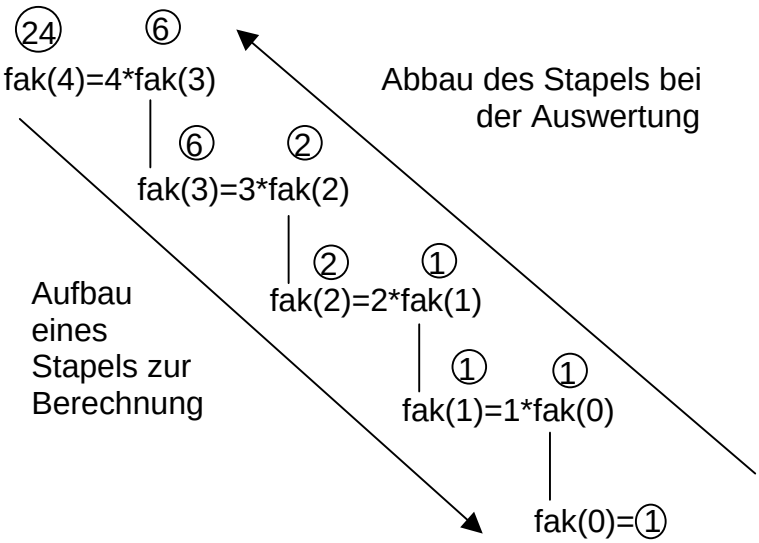
(siehe frühere iterative Lösung bei der For-Schleife)

Rekursionsvorschrift: $n! = n \cdot (n - 1)!$

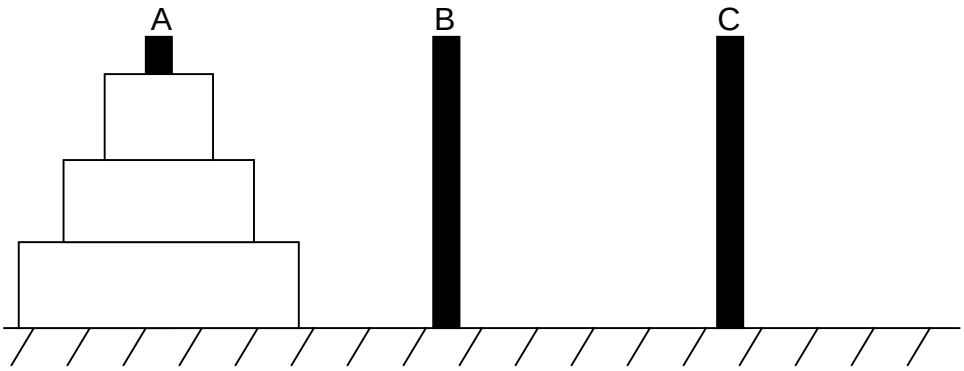
```
#include <iostream.h>
long fak(long n)
{
long erg;
if (n==0)
{
erg=1;
}
else
{
erg=n*fak(n-1);
}
return erg;
}
int main()
{
long n, ergeb;
cout<<"Bitte natürliche Zahl eingeben!";
cin>>n;
ergeb=fak(n);
cout<<"Ergebnis ist: "<<ergeb;
return 0;
}
```

Berechnungsbeispiel:

n = 4



Türme von Hanoi



Auf Platz A sind n Scheiben der Größe nach gestapelt. Sie sollen auf Platz C mit Hilfe von Platz B umgestapelt werden, wobei

1) immer nur eine Scheibe bewegt werden darf.

2) nie eine größere über einer kleineren Scheibe liegen darf.

Beispiele:

n = 1: $A \rightarrow C$ } 1 Aktion

n = 2: $A \rightarrow B$
$A \rightarrow C$
$B \rightarrow C$ } 3 Aktionen

n = 3: $A \rightarrow C$
$A \rightarrow B$
$C \rightarrow B$
$A \rightarrow C$
$B \rightarrow A$
$B \rightarrow C$
$A \rightarrow C$ } 7 Aktionen

allgemein: Anzahl der Bewegungen: $2^n - 1$

Rekursionsverfahren:

Annahme: Problem ist für (n-1) Scheiben gelöst.

1) Bringe die (oberen) (n-1) Scheiben von A nach B mit Hilfe von C.

2) Bewege die unterste Scheibe von A nach C.

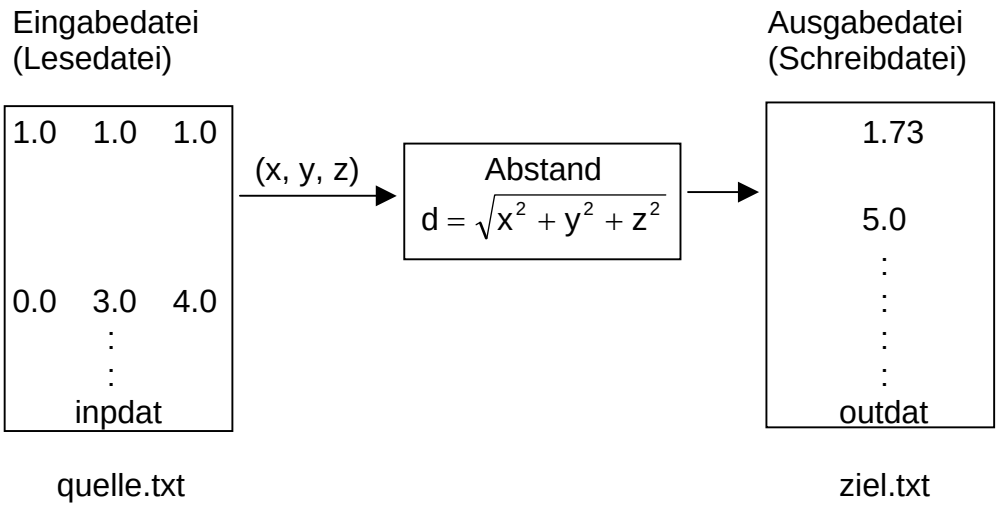
3) Bringe die (oberen) (n-1) Scheiben von B nach C mit Hilfe von A.

```
#include <iostream.h>
void bewegen(int n, char start, char ziel, char zwischen)
{
if (n==1)
{
cout<<"Bewege Scheibe von "<<start<<" nach "<<ziel<<endl;
}
else
{
bewegen(n-1, start, zwischen, ziel);
cout<<"Bewege Scheibe von "<<start<<" nach "<<ziel<<endl;
bewegen(n-1, zwischen, ziel, start);
}
}

int main()
{
int n;
cout<<"Geben Sie die Anzahl der Scheiben ein!";
cin>>n;
bewegen(n, 'A', 'C', 'B');
return 0;
}
```

8. Ein- und Ausgabe über Dateien

Aufgabe: Schreiben Sie ein Programm, das die Koordinaten (x, y, z) von n Punkten aus einer Datei liest, deren Abstände zum Ursprung berechnet und in einer Datei ablegt.



```
/*Dateientest*/
#include<fstream.h>
#include<math.h>
int main()
{
const int n=4;
ifstream inpdat;     //beliebige Namen
ofstream outdat;
float x, y, z, d;
inpdat.open("quelle.txt");
outdat.open("ziel.txt");
if(!inpdat)
{
cout<<"Lesedatei nicht vorhanden!";
return 1;
}
else
{
for(int i=1; i<=n; i++)
{
inpdat>>x>>y>>z;          //Lesen aus Datei inpdat
d=sqrt(x*x+y*y+z*z);
outdat<<d<<endl;          //Schreiben auf Datei outdat
}
inpdat.close();
outdat.close();
return 0;
}
}
```

Bemerkungen

1) Headerdatei <fstream.h>
- enthält alle notwendigen Definitionen und Funktionen für die Ein- / Ausgabe über Dateien
- muss zu Beginn des Programms eingebunden werden
- umfasst auch die Headerdatei <iostream.h>, die nicht mehr zusätzlich eingebunden werden muss

2) Dateitypen
- Eingabedateien sind vom Typ ifstream (input-file-stream)
- Ausgabedateien sind vom Typ ofstream (output-file-stream)

3) Interne und externe Dateinamen

interner Dateiname (z.B. inpdat, outdat):
- nur im vorliegenden Programm benutzt und bekannt
- im Programm wird Datei nur über den internen Namen angesprochen
- Name kann beliebig gewählt werden

externer Dateiname:
- Name der physikalischen Datei auf Betriebssystemebene
- umfasst – wenn notwendig – die gesamte Pfadstruktur, z.B. C:\windows\daten\quelle.txt

4) Öffnen der Dateien

"interner Dateiname"**.open**("externer Dateiname");
- Jede Datei, von der gelesen bzw. auf die geschrieben wird, muss geöffnet werden.
- Open-Befehl verknüpft internen mit externem Dateinamen.
- Existieren die externen Dateien nicht, so gibt es keine "automatischen" Fehlermeldungen. → zusätzliche Abfrage im Programm **if(!inpdat)**...
- Existiert eine Ausgabedatei nicht, so wird sie vom System automatisch angelegt.

5) Schließen von Dateien

"interner Dateiname"**.close**();
- Am Programmende werden alle Dateien automatisch geschlossen. (close ist nicht unbedingt notwendig)
- Vorteil des expliziten Schließens: Daten werden sofort abgespeichert und können bei nachfolgendem Programmabsturz nicht verlorengehen.

6) Datenein- und ausgabe
- Eingabe: "interner Dateiname">>zahl (≡ cin>>...)
- Ausgabe: "interner Dateiname"<<zahl (≡ cout>>...)

7) Erzeugen der Textdateien
- Öffne im Programmeditor neue Datei.
- Trage Daten ein und speichere Datei unter dem externen Dateinamen.
- Dateien können auch mit beliebigem Textsystem erzeugt werden.