Informationsverarbeitung (Modul IV)

Zwei Vorlesungen mit Praktikum für BT / UV

Fachbereich MuV, Hochschule Furtwangen University
Prof. Dr. Stefan von Weber
Version 03 / 2010

Informationsverarbeitung 1 - Prozedurale Programmierung

Vorlesung mit Praktikum für BT / UV zum Modul Inf.-Verarb. (IV)

Fachbereich MuV, Hochschule Furtwangen, Prof. Dr. Stefan von Weber

1. Einführung	1. Introduction
1.1 Historie	1.1 History
1.2 Rechneraufbau	1.2 Construction of a computer
1.3 Speicheradressen	1.3 Memory addresses
1.4 Dualzahlen	1.4 Dual numbers
1.5 Datentypen hardwareseitig	1.5 Hardware data types
1.6 Datentypen softwareseitig	1.6 Software data types
1.7 Softwareschichten	1.7 Software layers
1.8 Gleichungen und Anweisungen	1.8 Equations and instructions
2.Einführung in C	2. Introduction to C
2.1 Compile - Link - Run	2.1 Compile - Link - Run
2.2 Geradeausprogramm	2.2 Straight forward program
2.3 Grunddatentypen	2.3 Basic data types
2.4 Ausdrücke, Wertzuweisungen	2.4 Expressions, assign statement
2.5 Operatoren	2.5 Operators
2.6 Standardfunktionen	2.6 Standard functions
2.7 Blöcke, Verzweigungen	2.7 Blocks and branches
2.8 Schleifen (while, for)	2.8 Loops (while, for)
2.9 Matrizen und Strukturen	2.9 Matrices and structures
2.10 Aufzählungstypen, Typdefinitionen	2.10 Enumeration types, type definitions
2.11 Zeiger	2.11 Pointers
2.12 Funktionen	2.12 Functions
2.13 Datenfiles	2.13 Data files
2.14 Zeichenkettenfunktionen (alt)	2.14 String functions (old type)
2.15 Speicherklasse, Initialisierung, Typumwandlungen	2.15 Storage class, initializing, type conversions
3. C++ mit MFC	3. C++ with MFC
3.1 Klassen	3.1 Classes
3.2 Aufbau einer Klasse	3.2 Construction of a class
3.3 Beispiel Klasse CNurEineZahl	3.3 Example class CNurEineZahl
3.4 Basiswissen C++	3.4 Basic knowledge to C++
4. Einführung in Java	4. Introduction to Java
4.1 Grundlagen von Java	4.1 Basics of Java
4.2 Beispiel Ticketautomat	4.2 Example Ticket automat
4.3 Beispiel Sinuskurve	4.3 Example Sinus graph
4.4 Numerische Lösung von DGLs	4.4 Numerical solution of ODEs

Literatur / References

Bruce Eckel: C++ Einführungskurs, McGraw Hill, 1990
Frank Budszuhn, Thomas Reichel: Visual C++ 6, Addison-Wesley, 1999
David J. Barnes, Michael Kölling: Objektorientierte Programmierung mit JAVA, eine praxisnahe Einführung mit BlueJ, Pearson, 2003

1. Einführung

1.1 Historie

1943 Maschinensprache, Befehle wurden per Hand im Hexadezimalcode verschlüsselt
1950 Assemblersprache, jetzt durfte man statt "4C" schon "ADD" (addiere) schreiben
1952 FORTRAN, jetzt durfte man schon Anweisungen, z.B. A=B*C+4.6 schreiben
1972 Sprache C A:=B*C+4.6;
1990 Visual C++, objektorientierter Programmierung, aus C entstanden
1995 Java, objektorientierte rein dynamische Programmierung, aus C++ entstanden

1.2 Rechneraufbau

Prozessor (CPU = Central Processing Unit): Daten laden, Daten speichern, Rechnen, Logik
Speicher (Memory): Benutzt für die gerade laufenden Programme und ihre Daten
Externe Speicher: Festplatte, CD, Stick, Speicherkarte, ...
Externe Geräte: Tastatur, Maus, Bildschirm, Drucker, Lautsprecher, Plotter, ...
Karten: Graphik, Sound, Netzwerk, Prozess-Input-Output, ...

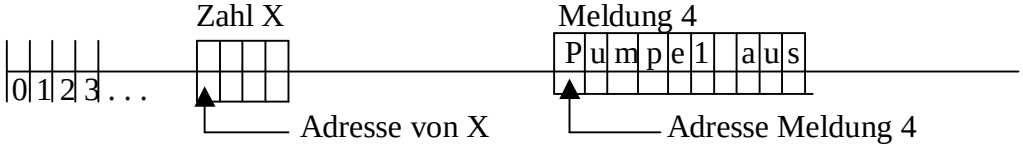
1.3 Speicheradressen

Der Arbeitsspeicher eines PC umfasst etwa 10^9 Byte = 1 Gigabyte oder kurz 1 GB. Damit ist nicht die Speicherkapazität der externen Speicher gemeint (Festplatte, CD, Stick, Speicherkarte, ...), denn die ist bedeutend höher (100 GB und größer).

Ein **Byte** besteht aus 8 Bit. Ein **Bit** kann lediglich eine Binärziffer 0 oder 1 speichern. Der Anwender kann nur mit Programmiertricks auf das einzelne Bit zugreifen. Die kleinste Speichereinheit für den Anwender ist das Byte. Es kann z.B. ein Zeichen im ASCII-Code aufnehmen (American Standard Code for Information Interchange). So hat der Buchstabe A die Bitbelegung 0100 0001.

Jedes der etwa 10^9 Bytes im Speicher hat eine Nummer 0, 1, 2, 3, ..., seine Speicheradresse. Die höchstmögliche Adresse ist bei 32-Bit Architektur 2^{32}-1 , das entspricht 4 GB.

Beispiel für eine Speicherbelegung mit einer Zahl und einem Text:



Aus Bytes bildet man Worte. Worte für Zahlen haben 2, 4 oder 8 Bytes. Die unterschiedlichen Register (Schnellspeicher) in der CPU haben ebenfalls diese Längen 2, 4, 8.

1.4 Dualzahlen

Rechnern arbeiten intern mit Dualzahlen. Auf dem Drucker oder auf dem Bildschirm mit Dezimalzahlen. Bei jeder Zahleneingabe und Zahlenausgabe werden die Zahlen deshalb konvertiert. Eine Dualzahl besteht nur aus den Ziffern 0 und 1.

Beispiel 1

107	=	64	+32		+8		+2	+1
		$\mathbf{1}\cdot 2^6$	$+\mathbf{1}\cdot 2^5$	$+\mathbf{0}\cdot 2^4$	$+\mathbf{1}\cdot 2^3$	$+\mathbf{0}\cdot 2^2$	$+\mathbf{1}\cdot 2^1$	$+\mathbf{1}\cdot 2^0$
		1	**1**	**0**	**1**	**0**	**1**	**1**

Hexadezimalzahlen dienen der verkürzten Schreibweise von Bitfolgen. Jeweils 4 Bit werden zu einem der 16 Zeichen 0, 1, 2, ..., A, B, ..., F zusammengefasst. (Mit 4 Bit lassen sich gerade die 16 Zahlen 0, 1, 2, ..., 15 darstellen.) Die folgende kleine Tabelle gibt die Umcodierung der 16 möglichen Bitfolgen in **Hexadezimalcode**:

0000 = 0	0001 = 1	0010 = 2	0011 = 3
0100 = 4	0101 = 5	0110 = 6	0111 = 7
1000 =8	1001 = 9	1010 = A	1011 = B
1100 = C	1101 = D	1110 = E	1111 = F

Setzt man vor die binäre Darstellung unserer Beispielzahl 107 noch eine binäre 0, dann ist

$$107_{\text{Dezimal}} \equiv 0\,1\,1\,0 \quad 1\,0\,1\,1_{\text{Binär}} \equiv 6\,B_{\text{Hexadezimal}}$$

Das 2-er-Komplement dient der Darstellung von negativen ganzen Dualzahlen. Beispiel sei die Zahl −107, die binär dargestellt werden soll:

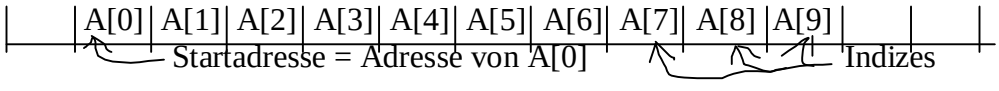
Wandle den Betrag 107 in eine Dualzahl	0000 0000 0110 1011
Negiere jedes Bit	1111 1111 1001 0100
Addiere nach den binären Rechenregeln eine 1	**1111 1111 1001 0101**

1.5 Datentypen hardwareseitig unterstützt (eine Auswahl)

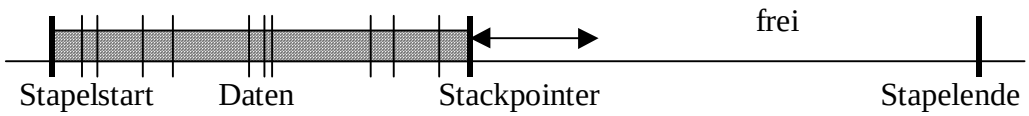
Zeichen	'A' '?'	'a'	'4'	Byte
Short Integer (kurze Ganzzahlen)	-32768	bis	+32767	2-Byte-Wort
Integer (normale Ganzzahlen)	$-2.1\cdot 10^9$	bis	$+2.1\cdot 10^9$	4-Byte-Wort
Float (kurze Gleitkommazahlen)	$\pm 8.4\cdot 10^{-37}$	bis	$\pm 3.4\cdot 10^{+38}$	4-Byte-Wort
Double (normale Gleitkommazahlen)	$\pm 4.2\cdot 10^{-307}$	bis	$\pm 1.7\cdot 10^{+308}$	8-Byte-Wort
Pointer (Zeiger, "Adresse von")	0	bis	$4.2\cdot 10^9$	4-Byte-Wort

1.6 Datentypen softwareseitig unterstützt (eine Auswahl)

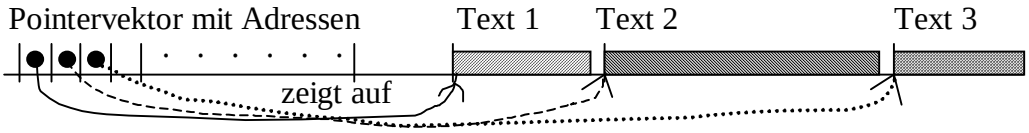
Vektor: Ist eine indizierte Folge von Vektorelementen gleicher Länge, z.B. A[10] ist in C ein Vektor mit 10 Elementen, die mit 0, 1, 2, ..., 9 indiziert sind.



Stapel (Stack, LiFo, Last in First out): Wird zum Zwischenspeichern kurzlebiger Informationen im Rechenprozess verwendet. Neue Daten werden immer am Stackpointer abgelegt, maximal bis zum Stapelende. Werden die Daten nicht mehr benötigt, wird vom Ende her der Speicher wieder freigegeben. Auf diese Weise pulsiert die Datenmenge im Stack ständig.

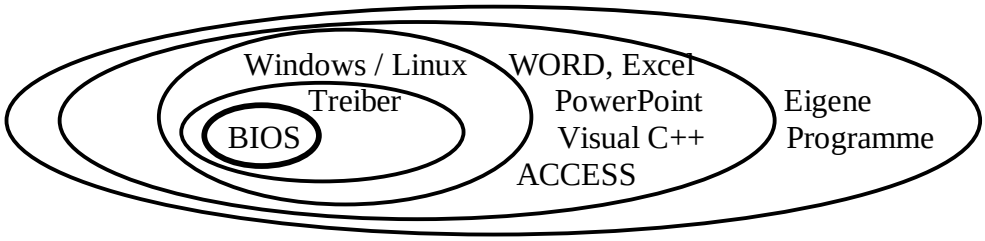


Listen werden immer dann benötigt, wenn extrem ungleich lange Informationseinheiten zu speichern und wiederzufinden sind, oder aber große Informationseinheiten zu sortieren sind. In letzterem Falle bewegt man beim Sortieren nicht die Daten selbst, sondern nur ihre Adressen. d.h. statt vieler Tausend Bytes nur 4 Bytes.



Heap (Haufen) ist ein Sammelspeicher, aus dem sich jedes Programm, das gerade läuft, mit Speicherplatz bedient. Mit einer Funktion *malloc* (Memory allocation) fordert das Programm vom Betriebssystem, z.B. von Windows, eine gewisse Speicherplatzmenge an. Wenn dieser Platz nicht mehr benötigt wird, gibt es mit Funktion *free* den Speicher an Windows zurück. Ein Problem dabei ist, dass der Heapspeicher immer mehr fragmentiert wird, d.h., dass die verfügbaren Stücken mit der Zeit immer kleiner werden. Beim Neustart von Windows ist aber alles wieder O.K.

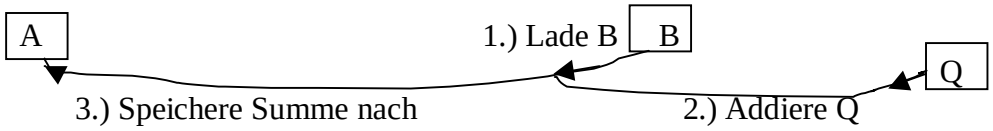
1.7 Softwareschichten



BIOS: Basic Input / Output System
Treiber: Vermittlerprogramme zwischen Windows und dem BIOS
Windows: Betriebssystem
Visual C++: Entwicklungsumgebung für C++-Programme

1.8 Der Unterschied zwischen Gleichung und Zuweisung

Mathematik	A = B + Q	statisch, ungerichtet, Seiten vertauschbar
Programmierung	A = B + Q ;	dynamisch, gerichtet, Seiten nicht vertauschbar



Auch die Reihenfolge der Zuweisungen muss in der Programmierung eingehalten werden. Die Mathematiker schreiben z.B. gerne

$$D = A^2 - \pi + \frac{A^3}{A-1} \qquad \text{mit} \qquad A = B + Q$$

In der Programmierung müssen alle Größen, die in einer Zuweisung auf der rechten Seite auftreten, einen Wert haben, d.h., der Programmierer muss hier zuerst den Wert von A berechnen und dann den Wert von D:

```
A = B + Q ;
D = A * A – 3.14159265 + A * A * A / (A – 1) ;
```

2 Einführung in C

2.1 Compile - Link - Run

1. Der Programmierer schreibt mit dem **Editor** (Textverarbeitungsprogramm) ein C-Programm. Der Vorgang sieht nicht viel anders aus, als wenn eine Sekretärin einen Brief an eine Firma schreibt.
2. Der **Präprozessor** (automatisches Redigierprogramm) verarbeitet alle Zeilen, die mit dem Zeichen # beginnen. Das sind einmal Konstantendefinitionen, wie #define PI 3.14159265 Der Präprozessor schneidet alle Auftreten von 'PI' im C-Programm heraus und ersetzt sie durch '3.14159265', macht also nichts weiter, als Professor Burlanger-Burr, der in seinen Reden das Wort 'Gott' durch die Worte 'das höhere Wesen, das wir verehren' ersetzt haben wollte (H. Böll, Dr. Murkes gesammeltes Schweigen). Weiterhin werden durch den Präprozessor Bibliotheksheader geladen, z.B. veranlasst die Anweisung #include <Mathe.h> den Präprozessor, die Header-Bibliothek der mathematischen Funktionen zu laden. Header sind nicht die Funktionen selbst, sondern nur die Kopfzeilen der Funktionen.
3. Der **C-Compiler** (Übersetzerprogramm) wandelt das redigierte C-Programm in Maschinenbefehle um (sogenannten Objektcode). Maschinenbefehle werden vom Prozessor des PC akzeptiert und ausgeführt. Dort, wo Funktionen aufgerufen werden, z.B. die Sinusfunktion, ist aber noch ein Fragezeichen, da die eigentliche Sinusberechnung ein extra Programm ist, das erst später vom Linker dazugeladen wird.
4. Der **Linker** sucht alle verlangten Funktionsprogramme aus den Programmbibliotheken zusammen und fügt sie und den Objektcode des C-Programms zu einem Exe-File zusammen.
5. Der **Lader** (Teil von Windows) stellt Speicherplatz bereit, lädt das Exe-File und startet es.
6. Das Exe-File ist die ausführbare Version des von uns geschriebenen C-Programms. Es verlangt z.B., dass wir Daten über die Tastatur eintippen und gibt uns berechnete Werte auf den Bildschirm aus.

Die Schritte 1-5 dauern auf dem PC je nach Programmlänge 3-15 Sekunden.

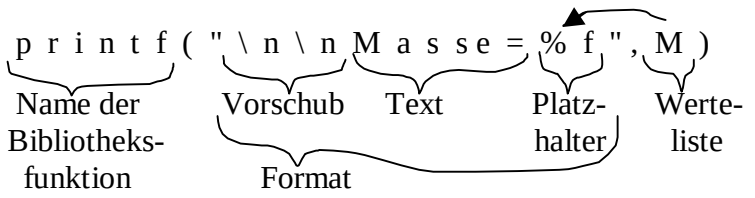
2.2 Ein Geradeausprogramm

Ein Geradeausprogramm hat keine Verzweigungen und keine Programmschleifen. Es ist die einfachste Form der Programmierung für einfache Aufgaben. Diese Sorte von Aufgaben könnte man aber ebenso gut oder besser mit Excel lösen.

Beispiel 2: Masse einer Kugel $M = \frac{4\pi}{3} \rho \cdot R^3$	1. Bibliotheken nennen 2. Konstanten definieren 3. Kopfzeile 4. Variablendeklarationen 5. Werte lesen 6. Berechnung 7. Werte ausgeben 8. Rückkehr zu Windows
#include <Stdio.h>	Standard-Input-Output Bibliothek
#include <Iostream.h>	Input-Output-Stream Bibliothek
#define PI 3.14159265	Z.B. Konstante PI definieren
	Leerzeile zur Auflockerung
// Masse einer Kugel	Kommentarzeile, Text beliebig
int main(.....)	Kopfzeile wird von Visual C++ geliefert
{	Blockbeginn des Programmkörpers
double R, rho, V, M ;	Variablendeklarationen
cout << "\nBitte R eintippen : ";	Zeilenvorschub und Textausgabe
cin >> R;	Zahleneingabe: Radius wird erwartet
cout<<"\nBitte rho eintippen : ";	Zeilenvorschub und Textausgabe
cin >> rho;	Zahleneingabe: Dichte wird erwartet
V = (4 * PI / 3) * R * R * R ;	Berechnung
M = V * rho ; // rho ist die Dichte	Berechnung (mit einem Kommentar)
printf ("\n\nVolumen=%f", V);	Formatierte Text und Zahlenausgabe
printf ("\n\nMasse=%f", M);	Formatierte Text und Zahlenausgabe
return (0);	Rückkehr zur Windows-C++-Umgebung
}	Blockende des Programmkörpers von Main()

Visual C++ liefert uns bereits ein fertiges kleines Programmgerüst, in dem schon eine Bibliothek *Stdafx* genannt wird (nicht löschen und keine Bibliothek davor setzen), die Main-Kopfzeile, die beiden Klammern und darin die Anweisung return (0); Die Main-Kopfzeile wird ebenfalls von Visual C++ richtig eingesetzt. Die sollten wir nicht ändern.

Das **%f** in printf("\n\nMasse=%f", M); ist ein Platzhalter im Ausgabeformat für eine auszugebende Gleitkommazahl vom Typ *float* oder *double*. Der duale Zahlenwert auf Speicherplatz M wird in eine lesbare Dezimalzahl konvertiert, d.h., in eine Folge von druckbaren Textzeichen, und diese werden an der Stelle %f eingesetzt.



Die Anweisung return (0); liefert an Windows ein Ergebnis zurück, einen sogenannten Rückkehrcode, der signalisiert, dass mit der Programmverarbeitung alles O.K. war.

/* Schreiben Sie viel Kommentar in ein Programm, das etwas komplizierter gebaut ist oder kompliziert gebaute Daten verarbeitet. Diese drei Zeilen sind ein Beispiel für einen mehrzeiligen Kommentar */
Einen 1-zeiligen Kommentar und einen Zeilenendekommentar sehen sie oben im Beispiel.

Übung 1:	**Body-Mass-Index** $K = G / H^2$ G=Gewicht in Kg, H=Körperhöhe in Meter Sie können mit der Stream-Funktion cout Text und Zahlenwert ausgeben, z.B. c o u t < < " \ n \ n I n d e x = " < < K ; Vorschub Text Zahl

2.3 Grunddatentypen (Variablen, Vektoren, Konstanten)

char Bst, Zeich, Vorname[50];	Deklaration von Zeichen- und Zeichenkettenvariablen
int i, nElems, N[10], ka;	Deklaration von Ganzzahlvariablen und -vektoren
double x, phi, A[10], karo_flaeche;	Deklaration von Gleitkommavariablen und Vektoren

Konstanten treten etwa in Zuweisungen, Formeln und Funktionsaufrufen auf. Definierte Konstanten (siehe Beispiel oben) werden durch ihren Namen vertreten, z.B. PI. Schreibt man die Zahl, das Zeichen oder die Zeichenkette direkt hin, heißt die Zeichenfolge Konstantenliteral. Beispiele mit Verwendung von Konstanten sind (Konstanten hier fett):

```
A = 4.926;
V = ( 4 * PI / 3 )*R*R*R;
```

Ganzzahlige Konstanten	0	4	17	43926452	
Gleitkommakonstanten	0.0	21.4	0.78	–3.14e-22	(lies $-3.14 \cdot 10^{-22}$)
Zeichenkonstanten	'0'	'A'	'b'	'*'	'''
	'\n'	Newline (Neue Zeile auf Bildschirm, Drucker, Datei)			
	'\0'	Maschinennull (binär die Bitfolge 0000 0000)			
	'\t'	Tabulatorzeichen in Datei- oder Druckausgaben			
	'\''	Das Apostroph ' selbst			
	'\\'	Der Backslash selbst			
Zeichenkettenkonstanten	"Bitte warten"			"A:\\DATA\\Mess.dat"	

2.4 Ausdrücke, Wertzuweisungen

x = 0 ;	Speichere Zahl 0 nach x (der alte Inhalt von x ist weg)
y = x + 4 ;	Speichere Summe x+4 nach y. x behält seinen Wert.
c = getchar( ) ;	Lies ein Zeichen von der Tastatur und speichere es nach c.
a = b = 1 ;	Speichere zuerst eine 1 nach b, speichere dann b nach a.
a = (b = 4) * 5 ;	Speichere zuerst 4 nach b, dann speichere b*5 nach a.
x = (a > b) ;	Wenn a>b, dann speichere eine 1 nach x (1=logisch *wahr*) Wenn a≤b, dann speichere eine 0 nach x (0=logisch *falsch*)

Übung 2	Wir schreiben die beiden Formeln als C-Zuweisungen mit zulässigen Bezeichnungen und in der richtigen Reihenfolge. Potenzen ersetzen wir durch mehrmaliges Multiplizieren, d.h., wir müssen der Klammer in der linken Formel einen Namen geben, sie Berechnen und dann erst multiplizieren. $\Psi = \left(\frac{\alpha * -1}{K^2 \cdot \pi^2} \right)^3$ $\alpha^* = \frac{4\pi}{(1-z)(1+K^2)}$

2.5 Operatoren (nach fallender Priorität geordnet)

*** / %**	mal durch Rest einer Ganzzahldivision **3*7** **x / 4.1** **17 % 4** ergibt 1 **(Vorsicht!!! Ganzzahldivision 7/3 ergibt 2 3/4 ergibt 0)**
+ -	a+b a–b –3.9 –a*b
> >= < <= == !=	a>b x >= 2.5 3 < b y <= (a+b) k == 0 k **!**= 0 Vergleiche liefern entweder 1 (wahr) oder 0 (falsch)
! (Negation)	**!** x **!** (a>b) macht aus 1 (wahr) eine 0 (false) und umgekehrt
&&	(a>b) && (x<1) Logisches UND (Konjunktion) liefert nur 1 (wahr), wenn beide Operanden 1 sind, sonst 0 (falsch)
| |	(a>b) | | (x<1) Logisches ODER (Disjunktion) liefert nur 0 (falsch), wenn beide Operanden 0 sind, sonst 1 (wahr) Das Zeichen | müssen Sie bei vielen Tastaturen mit der Tastenkombination "AltGr" und " > < | " schreiben

2.6 Standardfunktionen (aus der Bibliothek <Math.h>)

Funktion	Beispiel
abs( )	k = abs(j); i = abs(k+3); Betrag einer Ganzzahlgröße
fabs( )	x = fabs(z*z – PI); Betrag einer Gleitkommagröße
sin( ), cos( ), tan( )	x = sin (PI * y); Alle Winkelfunktionen erwarten Argument in Radiant
asin(), acos, atan()	alfa=atan(y / x); Umkehrfunktionen liefern Winkel in Radiant if (x<0) alfa = alfa + PI; (Bei Winkeln im 2. und 3. Quadranten)
exp( )	z = exp (–x*x); wie $z = e^{-x*x}$ Exponentialfunktion
log( )	x = log (2 * PI); wie $x = \ln(2\pi)$ natürlicher Logarithmus
sqrt( )	F = sqrt(a*a +1); wie $F = \sqrt{a^2+1}$ Quadratwurzel

Übung 3:	Wir schreiben die Formeln als C-Anweisungen mit den richtigen Funktionsnamen $\rho = \sqrt[4]{k^2 - \sin(x)}$ $m = \frac{m_0}{\sqrt{1 - v^2/c^2}}$ $\Psi^* = \frac{e^{k \cdot \sin(x)}}{\sqrt{1 - \lvert x^2 - y^2 \rvert}}$

2.7 Blöcke, Verzweigungen

{ ..... }
{ int a, b; double x, y; ..... } sind Blöcke

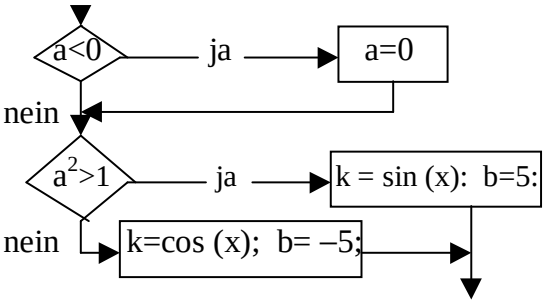
Die in einem Block deklarierten Variablen sind nur dort gültig.

if-Verzweigungen gestatten je nach Daten unterschiedliche Programmzweige zu durchlaufen. Die beiden möglichen Formen einer if-Verzweigung sind if-mit-nur-Ja-Zweig und if-mit-Ja-und-mit-Nein-Zweig.

if (Frage) Ja_Zweig; oder **if (Frage) { Ja-Zweig ; }**

if (Frage) { Ja_Zweig ; } else { Nein_Zweig ; }

Beispiel 3: Einen Pogrammflussplan mit Verzweigungen zeigt die nächste Graphik:

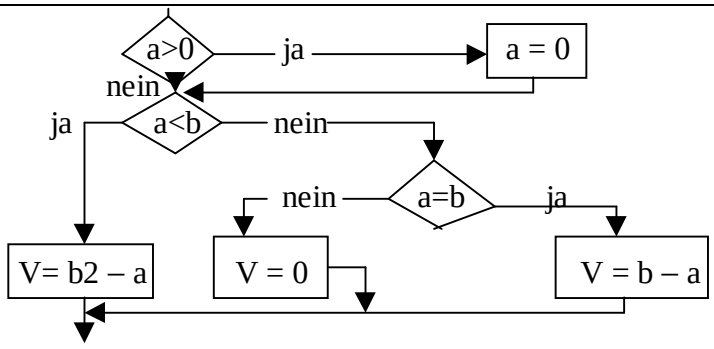


Das C-Programm dazu ist:

if (a < 0) a = 0; // Ist nur eine Anweisung im Zweig, kann man { } weglassen. Aber **if (a < 0) { a = 0; }** ist ebenso richtig
if (a*a > 1) { k = sin(x); b=5; } // Der Ja-Zweig kommt nach der Frage, dann else, dann
else { k = cos(x); b= -5; } der Nein-Zweig

Übung 4:

Programmieren Sie diesen Programmflussplan. Verwenden Sie unbedingt die geschweiften Klammern.



2.8 Schleifen

Schleifen wiederholen immer wieder ein Programmstück, z.B.

- addiere ein Vektorelement zu einer Summe, bis alle Elemente addiert sind
- Löse iterativ eine Gleichung, bis sie genau genug gelöst ist.
- Die **for-Schleife** nimmt man, wenn die Zahl der Durchläufe bekannt ist.
- Die **while-Schleife** nimmt man, wenn eine logische Bedingung die Entscheidung für einen erneuten Schleifendurchlauf liefert.

Sowohl die for-Schleife als auch die while-Schleife testen vor dem Eintritt in den Schleifenkörper, ob er durchlaufen werden soll. Es sind somit auch 0 Durchläufe möglich.

Die **do-while-Schleife** hingegen testet erst am Ende des ersten Durchlaufs, ob die Schleife wiederholt werden soll. Hier hat man immer mindestens einen Durchlauf.

2.8.1 while-Schleifen

while (Weitermachtest) Anweisung_oder_Block ;

Beispiel: Subtrahiere von der Variablen X immer wieder PI, bis X kleiner oder gleich PI ist. Ist X von vornherein kleiner oder gleich PI, dann wird die while-Schleife nicht durchlaufen.

```
while ( X>PI ) X = X - PI ;
```

Beispiel 4: Einlesen einer Messreihe. Der Nutzer signalisiert das Datenende durch die Eingabe einer sehr großen Zahl, z.B. 10^{25} (in C schreibt man einfach 1e25)

```
#include <Iostream.h>                  // Streambibliothek
#define  N  1000                       // Maximale Länge unserer Messreihe
int main(.....)                        // Vorgefertigte Kopfzeile des Main-Programms
{ int n;                               // n zählt die eingetippten Messwerte
  double z[N], zahl;                   // Vektor z speichert die maximal N Datenwerte

  n=0;                                 // Zähler ist anfangs 0
  do{                                  // Start einer do-while-Schleife
      cout<<"\nBitte Zahl eintippen : ";// Textausgabe: Aufforderung zum Eintippen
      cin>>zahl;                       // Lesen der eingetippten Zahl

      if(zahl<1e10)                    // Ist die Zahl für einen Messwert klein genug?
        { z[n]=zahl;                   // Im Ja-Fall speichere die Zahl im Vektor
          n=n+1;                       //                und erhöhe den Zähler um 1
        }                              // Ende der if-Verzweigung
     } while ( zahl < 1e10 );          // Weitermachtest (Schleife wiederholen?)

  cout<<"\nSie haben "<<n<<" Messwerte";   // Zur Information für den Anwender

  return ( 0 );                        // Rückkehr zu Visual C++
}                                      // Programmtextende
```

Übung 5:	**Iterative Berechnung der Wurzel w = $\sqrt{x}$** nach der Näherungsformel $w_{neu} = (w_{alt} + (x / w_{alt})) \cdot 0{,}5$ Wir beenden die Iteration, wenn $\left\|w_{neu}^2 - x\right\| \le 10^{-9}$ Startwert der Iteration ist $w_{alt}=1$. Nach jedem Durchlauf setzen wir einfach $w_{alt} = w_{neu}$ und rechnen weiter. Testen Sie das Verfahren zuerst mit dem Taschenrechner am Beispiel w=$\sqrt{4}$ auf einem Blatt Papier. **Vorsicht bei $\sqrt{1}$!!!**

2.8.2 for-Schleife

```
int  i , n ;
.........
f o r (   i = 0 ;  i < n ;   i++ )  Anweisung_oder_Block
```

Zähler nach jedem Durchlauf um 1 erhöhen
kein Durchlauf (mehr) bei Antwort "Nein"
Startwert des Schleifenzählers setzen

Die Anweisung bzw. der Block von Anweisungen wird hier n-mal durchlaufen. Falls n≤0 ist, wird die Schleife nicht durchlaufen und wirkt insgesamt lediglich wie i = 0 ;.

Beispiel 5: Mittelwert und Maximum der Messreihe mit n Zahlen, die im vorangegangenen Beispiel 4 gelesen wurde.

```
  .....                                  // Das vorangegangene Beispiel bis Info-Ausgabe
  cout<<"\nSie haben "<<n<<" Messwerte";

  int i;                                 // Zähler und Index der Vektorelemente
  double sz, zmittel, zmax;              // Variablendeklaration für Summe, Mittel, Max.

  sz = 0 ;                               // Vor Summation immer die Summe löschen
  zmax = -1e25;                          // Startbelegung für Maximumsuche ist -10^25

  for ( i=0 ; i<n ; i++ )                // Index muss mit 0 starten wegen z[0], z[1], ...
  {   sz = sz + z[i] ;                   // Addiere zur Summe Vektorelement z[i]
      if ( z[i]>zmax ) zmax=z[i];        // Tausche zmax aus, wenn z[i] größer ist
  }                                      // Schleifenende der for-Schleife

  zmittel = sz / n ;                     // Mittelwert = Summe / Anzahl

  cout<<"\n\nMittel     ="<<zmittel;     // Text- und Zahlenausgabe zum Mittelwert
  cout<<"\n\nMaximum="<<zmax;           // Dasselbe für das Maximum der z-Werte

  return ( 0 ) ;                         // Rückkehr zu Visual C++
}                                        // Programmtextende
```

Übung 6:	Berechnen Sie **Minimum** z_{min} und **Standardabweichung** σ_{n-1} der Messreihe $$\sigma_{n-1} = \sqrt{\frac{\left(\sum z_i^2\right) - n \cdot \bar{z}^2}{n-1}}$$ Dateneingabe und Berechnung des Mittelwertes aus den vorangegangenen Beispielen durch ..... andeuten.. Benutzen Sie ebenfalls die for-Schleife.

Übung 7:	Auf den Vektoren **A** und **B** seien jeweils n Zahlen abgespeichert. Berechnen Sie das **Skalarprodukt** $S = \sum_i^n A_i B_i$ und geben Sie den Wert aus.

2.8.3 Spezielle Operatoren (++ –– += *=)

C wurde von seinem Erfinder Richie auf extreme Kurzschreibweisen getrimmt.

i++	wirkt wie i = i + 1	Man benutzt es zum Erhöhen von Zählern
i--	wirkt wie i = i –1	Man benutzt es zum Erniedrigen von Zählern
S += A[i]	wirkt wie S = S + A[i]	Addiere zur Summe S einen Wert und speichere das neue S auf seinen alten Platz zurück
P *= A[i]	wirkt wie P = P * A[i]	Multipliziere P mit einem Wert und speichere das neue P auf seinen alten Platz zurück

Wichtig ! !	Summationsvariablen vor der Summation Null setzen, z.B.	**S = 0 ;**
	Produktbildungsvariablen vor der Produktbildung Eins setzen	**P = 1 ;**

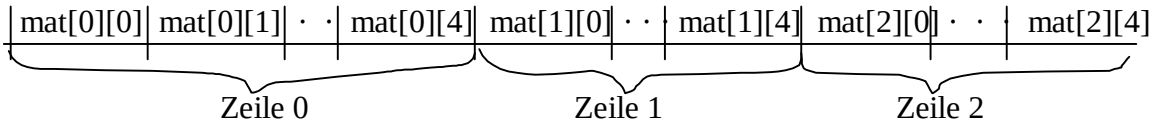
2.9 Matrizen und Strukturen

Matrizen sind 2-, 3-, ..., d-dimensionale Felder aus gleich langen Elementen desselben Datentyps. Der Sinn einer Matrix ist, dass der Computer mit einer simplen Formel ausrechnen kann, wo sich das Element (i, j) bzw. (i, j, k) auf dem Speicher befindet, und der PC auf diese Weise in wenigen Nanosekunden auf das Element zugreifen kann.

Beispiel 6: Deklaration einer Matrix mit 3 Zeilen und 5 Spalten. Elemente sind hier im Beispiel Zahlen vom Typ double.

double mat[3][5];

Die Anordnung der 15 Elemente auf dem Speicher erfolgt zeilenweise.

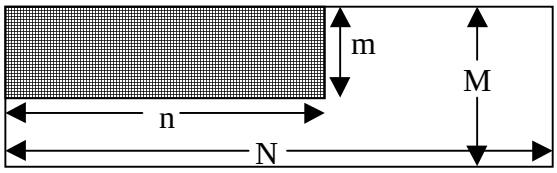


Beispiel 7: In C und C++ definiert man Matrizen fester Größe

```
#define   M   50          // Für die zu deklarierende Zeilenzahl der Matrix
#define   N   100         // Für die zu deklarierende Spaltenzahl der Matrix
.......                   // Programmkopf, andere Deklarationen
double  A[M][N];          // Matrixdeklaration mit M Zeilen und N Spalten
int  i, j,   m, n ;       // Indexvariablen und tatsächliche Zeilen-/ Spaltenzahl
.....                     // andere Deklarationen oder Programmzeilen
```

// Die ersten m Zeilen und n Spalten der Matrix A sollen z.B. mit Nullen belegt werden

```
for ( i=0 ; i<m ; i++ )                  // Wir verwenden zwei geschachtelte
{   for ( j=0 ; j<n ; j++ )   A[i][j] = 0 ;   }   // for-Schleifen
```

Meist werden von den Programmanwendern nur die ersten $m \le M$ Zeilen und $n \le N$ Spalten einer Matrix wirklich benutz. Der Rest der Matrix bleibt dann unbelegt.	

Strukturen (nur informativ) enthalten Daten unterschiedlichen Typs (und sind damit die Vorläufer der *Klassen* aus C++).

Beispiel 8: Beispiel für eine reine Strukturendefinition noch ohne definierte Objekte ist:

```
struct   patient  {  int      pat_num ;        // Patientennummer als Ganzzahl
                     char    name[50];        // Name als Zeichenkette maximale Länge 50
                     double gewicht;          // Gewicht, z.b. 54,7  als Gleitkommazahl
                  }
```

Mit der deklarierten Struktur können wir jetzt beliebig viele Objekte auf dem Speicher deklarieren, z.B. für einen Musterpatienten oder einen Vektor von Patientendaten mit 10.000 Plätzen

struct patient   musterpatient, pat_datei [10000];

2.10 Aufzählungstypen und Typdefinitionen (nur informativ)

Beispiel 9: Aufzählungstypen sind vereinfachte und komprimierte Konstantendefinitionen in Verbindung mit einer Typdefinition. Im Beispiel werden die Wochentage als Konstantennamen behandelt und jedem Namen wird ein Zahlenwert zugewiesen: Montag=0, Dienstag=1, ..., Sonntag=6. Gleichzeitig wird ein Typ **enum Tag** für Werte definiert, die nur die Zahlen 0–6 annehmen können.

enum   Tag (Montag, Dienstag, Mittwoch, Donnerstag, Freitag, Samstag, Sonntag);

enum Tag   gestern, heute, morgen; // deklariert 3 Variable vom Typ *enum Tag*

if (morgen == Samstag) .... wirkt dann wie **if (morgen == 5) ....**

Andere Typendefinitionen: Ein FORTRAN-Fan, den das Schicksal zur Programmiersprache C++ verschlagen hat, möchte unbedingt die Variablen i, j, k, l, m, n nicht unter der Typbezeichnung **int** deklarieren, sondern unter der Typbezeichnung **INDEX**.

```
typdef   int   INDEX;          // Index wird als gleichwertig mit int deklariert
INDEX   i, j, k, l, m, n ;     //  Wirkt wie   int  i, j, k, l, m, n ;
```

2.11 Pointer (Zeiger)

Beispiel 10: Pointer oder Zeiger sind Adressenvariablen. Eine solche Variable enthält die Speicheradresse eines Objekts. Adressen haben immer dieselbe Länge (32 Bit = 4 Byte).

Damit der Programmierer jedoch nicht in zu viele Fallen tappt, wird zu jeder Adresse gesagt, auf welche Objekttypen sie zeigen darf.

int x, y, *pj ; // deklariert zwei gewöhnliche Integervariable und einen *Zeiger auf Integer*. Zeiger pj darf also nur auf Integergrößen zeigen (z.B. auf x oder y.)

x = 10 ; // Eine normale Zuweisung: Integervariable x wird mit Wert 10 belegt
pj = &x ; // Zeiger pj wird mit Adresse von x belegt (Für **&** lies *Adresse von*)
y = *pj – 1 ; // wirkt hier wie y = x –1 ;
***pj = 3114 ;** // wirkt hier wie x = 3114 ;

2.12 Funktionen

Funktionen sind eigenständige Programme, die vom Main-Programm oder aus anderen Funktionen heraus aufgerufen werden können, und die etwas berechnen oder etwas steuern. Funktionen können über ihren Namen einen einzelnen Wert zurückliefern. Das kann, wie beim Sinus oder der Wurzelfunktion, der Funktionswert sein, es kann aber auch nur ein Rückkehrcode sein, d.h. eine Meldung, ob die Funktion korrekt oder nicht korrekt gearbeitet hat. Eine Funktion kann aber auch nichts zurückgeben.

Beginnt die Funktionsdefinition mit
void Name (Parameterliste), dann liefert die Funktion keinen Wert über den Namen zurück
int Name (Parameterliste), dann liefert die Funktion einen Integer-Wert über den Namen
double Name (Parameterliste), dann liefert die Funktion einen Double-Wert

Beispiel 11: Funktionsdefinition am Beispiel der Quadratwurzel

```
double Wurzel ( double x )                  // Kopfzeile mit Funktionstyp und Argumenttyp
{ double w_alt, w_neu;                      // Zwei lokale double Variable
  if ( x > 0 )                              // Verzweigung für den Fall, dass x≤0 ist
  {   w_alt = 1 ;                           // Startwert für die iterative Berechnung

      while( fabs ( w_alt * w_alt  –  x ) > 1e–9 )   // while-Schleife Weitermachbedingung
      {   w_neu = ( w_alt + x / w_alt ) * 0.5;       // iterative Näherungsformel
          w_alt = w_neu;                             // neues w_alt für die nächste Iteration
       }                                             // Schleifenkörperende
       return ( w_alt );                             // Funktionswert zurückgeben
  } else { return ( 0.0 );  }               // Wertrückgabe im Neinzweig des if(x>0) ...
}                                           // Ende des Quelltextes der Funktion Wurzel
```

Beispiel 12: Parameterlisten

```
#define   M   100                           /        / Konstantendefinition
void f ( double x, double y, int n, double A[M] )     // Kopfzeile Funktionsdefinition
```

die Funktionsdefinition von Funktion f nennt 4 Parameter in der Parameterliste. Die Funktion erwartet bei einem Aufruf den

1. Parameter als Double-Größe
2. Parameter als Double-Größe
3. Parameter als Integer-Größe
4. Parameter als Double-Vektor mit M Elementen

Innerhalb der Funktion f heißen die Parameter x, y, n, A.

Wichtig ! ! Der Funktionsaufruf muss die Parameter in derselben Reihenfolge, mit denselben Typen und in derselben Anzahl liefern, wie die Funktion definiert wurde. Lediglich die Namen sind beim Aufruf beliebig. Im Gegenteil, die aktuellen Parameter des Funktionsaufrufs benötigen nicht einmal Namen, sondern es dürfen auch Formelausdrücke sein.

Wir wollen unsere selbst definierte Wurzelfunktion bei der Berechnung der folgenden Formel einsetzen: $X = \gamma \cdot \sqrt{1 + \sqrt{\pi \cdot y - 1}}$

Da die Formel 2 mal die Wurzel enthält, müssen wir auch in unserer Programmzeile 2 mal unsere selbst definierte Wurzelfunktion aufrufen:

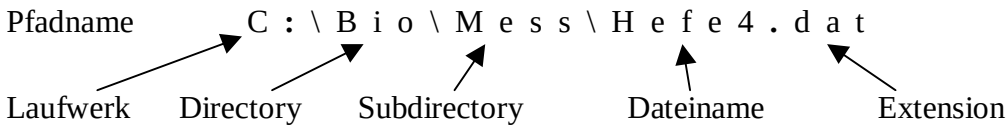
X = Gamma * Wurzel (1 . 0 + Wurzel (PI * y – 1 . 0)) ;

Zuerst wird die Klammer PI*y–1 berechnet, dann wird dieser Wert vom Typ *double* an die Funktion Wurzel abgeschickt. Die zieht aus dem Wert PI*y–1 die Wurzel und liefert ihn als Double-Zahl zurück. Jetzt wird eine 1 addiert und dieses neue Double-Zwischenresultat wird wieder an die Wurzelfunktion abgeschickt. Die zieht die Wurzel und liefert das Ergebnis. Das wird mit Gamma multipliziert und nach X gespeichert.

Übung 8a	Programmieren Sie die Funktion x! (Fakultaet). 0!=1, 1!=1, x!=1·2·3···x. Für Argumente x<0 bzw.x>100 soll die Funktion den Wert -1.0 zurückgeben. Das Argument x ist vom Typ *int*. Der Funktionswert ist vom Typ *double*. Verwenden Sie eine for-Schleife.
Übung 8b	Schreiben Sie ein kleines Programm, das die Anzahl N! der möglichen Anordnungen von N unterscheidbaren Objekten berechnet. Benutzen Sie dabei die in Übung 8a programmierte Funktion Fakultaet. N lesen Sie per Tastatur ein.

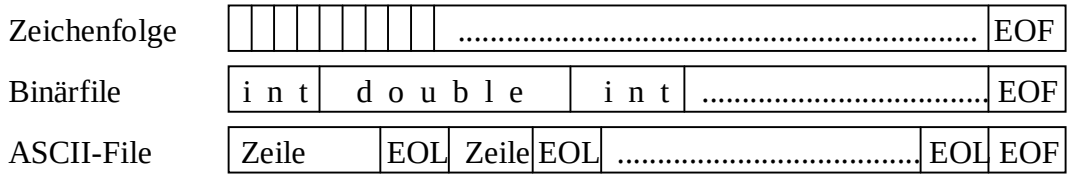
2.13 Datenfiles (Dateien auf Festplatten, Sticks, CDs, ...)

Jede Datei (File) hat einen Namen und steht auf einem Datenträger auf einem Laufwerk. Der Datenträger hat ein Inhaltsverzeichnis (Directory). Der Weg, wie man zur Datei gelangt, heißt Pfadname (Pathname).



Es gibt zahlreiche Dateiarten (Filearten). Die einfachste Datei ist die **Zeichenfolge**. Sie besteht aus einer beliebig langen Folge von Zeichen (Bytes) abgeschlossen durch ein spezielles Zeichen **EOF** (End-of-File). Dieses Zeichen steht übrigens am Ende jeder Datei. Ein **Binärfile** besteht aus aneinandergereihten Computerworten mit internen (binären) Zahlen

vom Typ int, double usw. und/oder Zeichenketten. Ein **ASCII-File** besteht nur aus druckbaren Zeichen in ASCII-Codierung, und ist in einzelne Zeilen gegliedert. Die Zeilen dürfen unterschiedlich lang sein. Jede Zeile endet mit dem speziellen Zeichen **EOL** (End-of-Line).



Dateiarbeit ist das Zusammenspiel einer ganzen Hierarchie von Programmen. Ganz unten steht das BIOS, das die physikalische Seite der Datenübertragung steuert. Darüber kommen die Gerätetreiber, die spezielle Eigenheiten eines Laufwerks berücksichtigen, z.B., dass bei Festplatten nur ganze Sektoren geschrieben werden können (Sektor = Abschnitt auf der Magnetspur auf der Festplatte). Darüber kommt das Betriebssystem, z.B. Windows, das Pufferspeicher bereitstellt. **Pufferspeicher** sind doppelt angelegte Datenspeicher auf dem Arbeitsspeicher, die beim Öffnen (open) einer Datei mit Daten aus der Datei gefüllt werden. Von dort liest man die Daten mit einer C-Eingabefunktion. Ist ein Puffer ausgelesen, schaltet Windows automatisch auf den anderen Puffer um und füllt gleichzeitig den ersten Puffer wieder mit neuen Daten auf. Bei einer Ausgabedatei wird erst auf den Datenträger ausgegeben, wenn ein Puffer voll ist, oder aber die Datei beendet wird (close).

2.13.1 Files öffnen

1. Pfadnamen übermitteln. Dieser kann direkt als Literal gegeben werden, oder aber als Inhalt einer Zeichenkettenvariablen.
2. Festlegung, ob von der Datei gelesen werden soll („r“ = read), oder aber auf sie geschrieben werden soll („w“ = write) oder ob sie verlängert werden soll („a“ = append).

Das Betriebssystem macht zusätzlich

- eine interne Infotabelle zu jedem geöffneten File auf. Deren Nummer (Adresse) heißt Filehandle (Filepointer, Filenummer, Refnum). Dieses *Handle* ist für die Arbeit mit der geöffneten Datei wichtig.
- zwei Pufferspeicher auf
- und einen Check, ob das Laufwerk existiert, der Datenträger existiert, das Directory und eventuell die Subdirectories existieren, das File existiert und sich lesen lässt.

Beispiel 13:

```
FILE   *ein, *aus;                  // Zwei Filehandlevariablen (Filepointervariablen)
char   auspfadname [50];            // Zeichenkettenvariable, z.B. für einen Pfadnamen
.........
cout<<"\nBitte Ausgabepfadnamen eintippen : ";  // Der Pfadname wird über die Tastatur
cin>>auspfadname;                               // eingetippt und übernommen
.........
ein = fopen ( " C:\\Bio\\Hefe7.dat", "r" );     // Dateieröffnung für Lesen
aus = fopen ( auspfadname, "w" );               // Dateieröffnung für Ausgabe
```

Bei der Eingabe-Datei ist der Pfadname z.B. als Literal fest vorgegeben. Beachten Sie, dass in Zeichenkettenliteralen der Backslash das Zeichen \\ ist, d.h. dass der Backslash doppelt

geschrieben werden muss. Bei der Ausgabedatei wird der Pfadname z.B. über eine Zeichenkettenvariable übergeben. Natürlich muss man dieser Variablen vorher einen Wert geben, nämlich einen gültigen Pfadnamen (wurde über *cout* und *cin* im Beispiel realisiert). Jeder Fehler im eingetippten Pfadnamen führt zum Abbruch des Programms. (Werden Pfadnamen zur Laufzeit des Programms über die Tastatur eingetippt, dann dürfen Sie den Backslash nicht doppelt schreiben. Er ist dann ein normales lesbares Zeichen.)

2.13.2 Files schließen

Beim Schließen einer Datei macht das Betriebssystem folgendes:
1. Bei Ausgabedateien werden die letzten Daten aus dem Puffer ausgegeben
2. Die Pufferspeicher und die Infotabelle werden freigegeben, d.h., der Speicher wird an Windows zurückgegeben.

```
fclose(ein);        // Schließt unsere Eingabedatei
fclose(aus);        // Schließt unsere Ausgabedatei
```

2.13.3 Unformatierte Ein- und Ausgabe von Zeichenketten (Zeilen)

Beispiel 14: Lies alle Sätze eines ASCII-Files und gib sie auf 10 Zeichen gekürzt wieder aus.

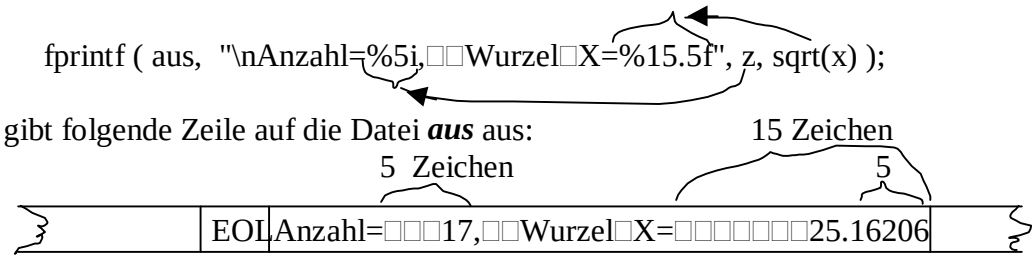
```
#include <Stdio.h>                      // Standard-Input-Output Bibliothek
int main(.............)                 // Kopfzeile des Main-Programms
{  FILE  *ein, *aus;                    // Blockanfang mit Deklaration der Filehandles
   char text[300], *status;             // Deklaration Textvariable, Status-Variable

   ein = fopen( "E:\\Mantras.txt", "r" );  // Datei Mantras auf Laufwerk E öffnen
   aus = fopen( "A:\\Leben.txt", "w" );    // Datei Leben auf Laufwerk A öffnen

// Start einer while-Schleife. Bei jedem Durchlauf wird als Erstes eine Zeile der
// Eingabedatei gelesen. Der Schleifenkörper selbst wird aber nur durchlaufen, wenn
// der Rückkehrcode der fget-Funktion ungleich NULL ist. (NULL ist eine spezielle
// Konstante, die in der Bibliothek Stdio.h definiert ist. Sie signalisiert hier das Dateiende.

  while(( status == fgets(text, 300, ein)) != NULL )
  {  text[10]='\n';                     // Speichere das Zeichen Newline in die Zeile
     text[11]='\0';                     // Speichere Ketten-Ende-Zeichen in die Zeile
     fputs(text, aus);                  // Gib die gekürzte Zeichenkette aus.
  }                                     // Ende der while-Schleife

  fclose (ein);                         // Schließen der Dateien
  fclose(aus);
  return (0);                           // Rückkehrcode und Rückkehr zu Windows
}                                       // Ende von Main
```

2.13.4 Formatierte Ausgabe / Eingabe

Die formatierte Ausgabe in C ist harmlos. Die formatierte Eingabe in C ist tückisch. Eigentlich kann man nur formatiert lesen, wenn die Daten mit exakt demselben Format auch

ausgegeben wurden, d.h., man kann eigentlich nur Daten formatiert lesen, die man auch selbst formatiert geschrieben hat.

Beispiel 15: In unserem Beispiel werden folgende Variablen mit den folgenden Werten verwendet: z = 17, x = 633.129, $\sqrt{x}$ =25.162055 (Das Zeichen □ steht hier in den Beispielen für das Leerzeichen (Blank, Leertaste), damit man die Blanks zählen kann)



Das Ausgabeformat enthält einen Vorschub (\n), einen Text (Anzahl=), einen Platzhalter (%5i) für eine Ganzzahl mit 5 Druckstellen, die rechtsbündig ausgefüllt werden, wieder Text(**,** Wurzel X=), und einen zweiten Platzhalter (%15.5f), diesmal für Gleitkommazahlen mit insgesamt 15 Druckstellen, davon 5 Dezimalstellen. Auch diese 15 Druckstellen werden rechtsbündig aufgefüllt. Die letzte Dezimalstelle wird gerundet. Die Ausgabeliste (z, sqrt(x)) muss so viele Werte in der richtigen Reihenfolge und mit dem richtigen Typ liefern, wie Platzhalter im Format sind. Variable z ist eine Integervariable und bedient den i-Platzhalter, sqrt(x) ist eine Formel, liefert einen Double-Wert und bedient den f-Platzhalter.

Mit der Eingabeanweisung

fscanf (ein, "\nAnzahl=%5i, □□Wurzel □X=%15. 5f", &z, &wx);

könnten wir die beiden Zahlen wieder von der Datei lesen. Das gelingt jedoch mit keinem anderen Format.

Ganz wichtig bei Eingabelisten ist, dass diese nur Adressen von Variablen oder von Vektor- bzw. Matrixelementen enthalten dürfen. **&z** ist die Adresse von z, **&wx** ist die Adresse einer Variablen wx. (Zur Erinnerung: Die Angabe eines Vektorelements, z.B. **A[5]** oder **A[k]** , in einer Eingabeliste ist bereits eine Adressangabe, und muss so stehen bleiben ohne das Zeichen & davor.)

Übung 9:	Ausgabe der Tabelle der Heuberg-Häberlein-Funktion $HH(\eta)=\sqrt{1+\sin(\eta)}\ e^{1+\ln(\eta)}$ Die Ausgabe erfordert u.a. einen e-Platzhalter im Ausgabeformat und soll etwa in der folgenden Form erfolgen: Tabelle der Heuberg-Häberlein-Funktion Eta HH(Eta) 0.1 ±1.2345678e±123 ... . 5.0 .....

2.14 Einige Zeichenkettenfunktionen (alten Typs) aus <String.h>

In C++ gibt es eine alte und eine neue Form der Zeichenketten. Die alte Form wird weiterbenutzt, da allzu viele Bibliotheksprogramme sie noch benötigen und man nicht alle umschreiben möchte. Die neue Form, die nur in C++ und Java verfügbar ist, benutzt den Klassennamen *String* für die Deklaration, und ist eigentlich viel einfacher zu handhaben.

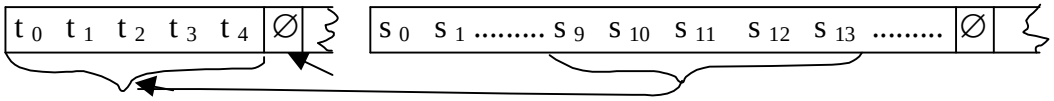
2.14.1 Beispiel 16: für die alte Form ist

```
char z[50];          0  1 2 3 4 5 6 7 8 ..... 49
                     W e b e r ∅
```

Die Zeichenkette z kann hier im Beispiel maximal 49 Zeichen aufnehmen. Das Zeichenkettenende wird durch das spezielle Zeichen ∅ markiert. Dieses Zeichen zählt beim Speicherplatz mit, nicht aber, wenn man die Länge der Zeichenkette (im Beispiel 5) bestimmt.

2.14.2 Beispiele 17: für alte Zeichenkettenfunktionen sind:

```
char Texte  [1000][20];                 // Deklariert Vektor von 20-er Zeichenketten
char s[100], t[100], s1[100], s2[100];  // Deklariert vier 100-er Zeichenkettenvariablen

strcpy ( s , "D:\\Bio\\Hefe7.dat" );    // Kopiert das Konstantenliteral "..." nach s

strcpy ( s1, s );                       // Kopiert Zeichenkette s nach s1

strcpy ( Texte[77], s );                // kopiert s auf Vektor Texte, Element 77

int k;                                  // speichert Länge von s nach k (∅ zählt nicht
k = strlen( s );                        // mit, wohl aber andere Sonderzeichen, z.B.
                                        // das EOL-Zeichen einer Eingabezeile
strncpy ( t, &s[9], 5 );                // Schneidet die Zeichen s[9] bis s[13] aus der
                                        // kette s und speichert sie nach t
```



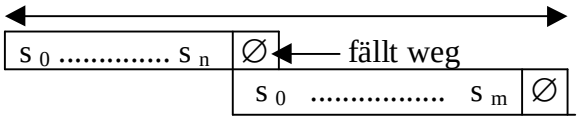
Dass man in diesem Beispiel "Adresse von s[9]" (&s[9]) schreiben muss, hängt mit der Funktion strncpy zusammen, die als zweites Argument die Adresse einer Zeichenkette verlangt. Würden wir nur s[9] schreiben, würde das Zeichen s[9] selbst an die Funktion geschickt werden, und die kann mit dem Zeichen nichts anfangen, da sie eine Adresse erwartet. (C sein Balla-Balla. Einfach nur lachen, nicht denken darüber nach.)

```
int i;                                  // Vergleicht alphanumerisch Kette s1 und s2
i = strcmp( s1 , s2 );                  // i bekommt einen Wert <0, wenn s1<s2 ist
                                        // i bekommt den Wert 0, wenn s1=s2 ist
                                        // i bekommt einen Wert >0, wenn s1>s2 ist
```

Beim alphanumerischen Vergleich ist z.B. "Otto" < "Ottolein", da Kette "Otto" kürzer ist.
"Atto" < "Otto", da A im Alphabet vor O kommt

strcat (s1, s2); // hängt Kette s2 ans Ende von s1

Concatinate Strings:



strncat (s1, &t[9], 5); // hängt 5 Zeichen aus t an das Ende von s1 dran

s_0 ..... s_n t_9 t_{10} t_{11} t_{12} t_{13} ∅

Wichtig!! Wenn nicht als Compiler-Option verlangt, dann prüft keine der Funktionen, ob die Resultatkette tatsächlich genügend Speicherplatz vorfindet. Im schlimmsten Falle werden wichtige Daten, die zufällig dahinter stehen, mit unseren Zeichen einfach überschrieben.

2.14.3 Einige Umwandlungsfunktionen

Beispiel 18: Umwandlung in Großbuchstaben (bzw. in Kleinbuchstaben)

```
int i, n;                          // Umwandlung aller Buchstaben in einer
n = strlen( s );                   // Zeichenkette in Großbuchstaben (bzw. mit
for ( i=0; i<n; i++) s[i]=toupper(s[i]);   // der Funktion tolower( ) in Kleinbuchstaben)
```

Beispiel 19: Umwandlung einer Zahlenkette in eine Internzahl

```
#include <Stdlib.h>                // Standardbibliothek
#define AUSFALL (–1e25)            // Ausfallwert als negative große Zahlkonstante
.....
double x;                          // Deklaration einer Double-Variablen
char zahlenkette[50], *pEnd;       // Deklaration einer Zeichenkettenvariablen und
                                   // einer Zeigervariablen auf Zeichenketten
cin>>zahlenkette;                  // Eingabe einer Zahl in Form einer Zeichenkette

x = strtod (zahlenkette, &pEnd );  // Umwandlung "String to Decimal", d.h. die
                                   // Zeichenkette wird in eine Internzahl gewandelt
```

// Der Zeiger pEnd enthält die Adresse des Zeichens, bei dem die Zahlenumwandlung stoppt.
// Falls die Zahlenkette eine gültige Zahl war, z.B. **3.14159265** , dann stoppt die
// Umwandlung genau nach der letzten Ziffer. War die Zahlenkette keine gültige Zahl, z.B.
// **3.14A59265** , dann stoppt die Umwandlung beim ersten nicht interpretierbaren Zeichen,
// hier also beim A. Mit einer if-Anweisung kann man nach der Umwandlung bei fehlerhafter
// Umwandlung irgendwie reagieren, z.B. nochmalige Eingabe verlangen, oder wie hier einen
// Platzhalter AUSFALL einsetzen. Die späteren Programmteile sollten aber mit Ausfällen
// besser nicht rechnen.

if ((pEnd – zahlenkette) != strlen(zahlenkette)) x = AUSFALL;

Wenn die Differenz aus Stoppadresse *pEnd* und Anfangsadresse *zahlenkette* ungleich der Länge der Zahlenkette ist, dann war etwas faul bei der Umwandlung. Wir setzen in diesem Fall x = AUSFALL. War alles O.K., wirkt die if-Anweisung wie eine Leeranweisung, und Variable x behält ihren Wert aus der Zahlumwandlung strtod( ).

2.14.4 Beispiel 20: Kaskadendaten in Tabellenform bringen

Ein Datenlogger registriert alle 5 Sekunden 3 Werte G1, G2, G3. Die Zahlen stehen als einzelne Zeilen einer ASCII-Datei untereinander. Werte 0 und – – – sind Ausfälle, die in der tabellarischen Form durch den Ausfallwert -10^{25} dargestellt werden sollen. Es wird vorausgesetzt, dass immer komplette 3-er-Gruppen an Zahlen vorliegen. Die Zahlen des Datenloggers haben ein Dezimalkomma (keinen Punkt, wie C das erwartet).

A:\UVPRaktikum\Kaskade.dat	——>	C:\UV\Kaskade.dat			
43,294		t	G1	G2	G3
41,468		0	±1.234e+123	±1.234e+123	±1.234e+123
0 (ein Ausfall)		5	±1.234e+123	±1.234e+123	±1.234e+123
37,262		10	±1.234e+123	±1.234e+123	±1.234e+123
- - - (ein Ausfall)		..	....	....	....
....		..	....	....	....

```
#include <Stdlib.h>                 // Standardbibliothek mit Funktion  strtod( )
#include <String.h>                 // Zeichenkettenfunktionen
#include <Stdio.h>                  // Standard-Input-Output fopen( ), fgets( ), ...

int main (............)             // Kopfzeile Main-Programm
{ int i, j, k, t, e:                // lokale Zähler, Indexgrößen
  double x, G[3];                   // Lokale Double-Variable, Double-Vektor
  char zahl[50], *pEnd;             // Zeichenkettenvariable, Pointer auf Zeichenkette
  FILE *ein , *aus;                 // Filepointervariablen

  ein = fopen( "A:\\UVPRaktikum\\Kaskade.dat", "r");   // Datei für Lesen öffnen
  aus = fopen("C:\\UV\\Kaskade.dat", "w");             // Datei für Schreiben öffnen

  fprintf ( aus, "  t   G1    G2    G3");   // Ausgabe der Kopfzeile der Tabelle
  t=0;                                      // Sekundenzähler

// Start einer while-Schleife, in der alle Zeilen der Eingabedatei nach und nach in 3-er-
// Gruppen gelesen werden

  while ( fgets(zahl, 50, ein) != NULL)     // Schleifenbeginn, Eingabe und Weitermachtest
  { for ( i=0; i<3; i++)                    // Zähler i zählt die Zahlen einer 3-er-Gruppe
    { if (i>0) fgets(zahl, 50, ein);        // G1 wurde oben gelesen, aber nicht G2, G3

      // Jede Zahl wird jetzt einzeln umgewandelt und dabei geprüft
      k=strlen(zahl);                       // Aus wieviel Zeichen besteht die Eingabezeile?
      k=k–1;                                // Das EOL-Zeichen wollen wir nicht mitzählen
      for ( j=0; j<k; j++)                  // Durchlaufe jedes Zeichen der Eingabezeile und
      { if ( zahl[j] == ',' ) zahl[j] = '.' ; }   // ersetze Kommas durch Punkte
      e = 0;                                // Setze unseren hauseigenen Fehlermelder 0
      x = strtod ( zahl, &pEnd);            // Wandle Zeichenkette in Internzahl um
      if ( ( pEnd – zahl ) != k ) e = 1;    // Bei Konvertierungsfehler e=1 setzen
      if ( (e==0) && (x==0) ) e=1;          // Kein Konvertierungsfehler, aber Zahl ist 0
      if ( e==0 ) G[i] = x;                 // Falls kein Fehler, dann x abspeichern, im
```

```
    else        G[i]= –1e25;          // Fehlerfalle jedoch unsere Ausfallcodierung
  }                                   // Ende der for-Schleife for (i=0; ....)

  // Eine 3-er-Gruppe von Zahlen wurde gelesen, geprüft und steht auf Vektor G bereit
  // und wird mit einer formatierten Ausgabe auf die Ausgabedatei geschrieben

  fprintf ( aus, "\n%5i%15. 4e%15. 4e%15. 4e", t, G[0], G[1], G[2] );

  t = t+5;                            // Erhöhung des Sekundenzählers
 }                                    // Ende der while-Schleife

// Die while-Schleife wird verlassen, wenn beim fgets( ) das File-Ende diagnostiziert wird

 fclose(ein);                         // Datei schließen
 fclose(ein);                         // Datei schließen

 return (0);                          // zurück zu Windows
}                                     // Ende von Main
```

Übung 10:	Eingabe einer Messreihe von Zahlen auf einen Vektor x über die Tastatur ähnlich dem Beispiel 4. Jede Zahl soll jedoch mit der Streamfunktion *cin* als Zeichenkette gelesen werden. Die Aufforderung zur Zahleneingabe, die Konvertierung und Prüfung setzen sie in eine while-Schleife, die innerhalb der while-Schleife liegt, die im Beispiel 4 schon vorhanden ist. Diese innere Schleife wird nur verlassen, wenn eine richtige Zahl oder das Wort *Ende* in beliebiger Groß-Klein-Schreibung eingegeben wurde. Bei falscher Eingabe geben Sie die Zeile *Fehlerhafte Eingabe. Wiederholung* aus. Benutzen Sie die Integervariablen *fehler* und *ende* als Merker, ob ein Fehler oder das Wort ENDE aufgetreten ist.

2.15 Speicherklasse, Initialisierung, Typumwandlungen (informativ)

Beispiel 21: Speicherklassen Die Speicherklasse bestimmt die "Lebensdauer" eines Wertes. Es gibt die Speicherklassen **extern, static** und **automatic**. Ein Wert wird **verdeckt**, wenn ein neuer, gleichnamiger Wert auf einem tieferen Blockungsniveau deklariert wird.

```
#include .....            // Zeilen noch vor dem ersten Programm
double alfa, x;           // extern: Überall verfügbar, wenn nicht verdeckt
int main(.....)
{ double A,y;             // automatic: Nur in Main verfügbar. Nur wenn nicht verdeckt
  .....}
double funk(double x);    // extern: Name funk überall verfügbar, wenn nicht verdeckt
                          // automatic: Größe x nur im Programm funk verfügbar, w.n.v.
{ double A, b. z;         // automatic: Das "A" in funk ist ein anderes A, als das in Main
                          // Bei Wiedereintritt in funk können die Werte gelöscht sein
  static double m[3];     // static: Bei Wiedereintritt in funk sind diese Werte noch da
  .....
  { double b, u;          // automatic: Variable b und u nur in diesem Block verfügbar
    .....                 // Das neue "b" verdeckt das alte "b" vom Funktionsanfang
}
```

Beispiel 22: Initialisierung. Initialisieren heißt eine Variable zu belegen. Das kann

- nur einmalig beim Laden des Programms,
- bei jedem Neustart des Programms,
- bei jedem Wiedereintritt in ein Programm oder einen Block

geschehen.

```
#include .....          // Zeilen noch vor dem ersten Programm
int x=1;                // extern: x wird einmalig beim Laden 1 gesetzt
#define MAX 512         // extern: MAX wird einmalig beim Laden 512 gesetzt
int j=MAX/2;            // extern: j wird einmalig beim Laden 512 / 2 = 256 gesetzt
char Puffer[MAX];       // extern: Puffer wird einmalig beim Laden mit Nullen belegt
char *first=Puffer;     // extern: Pointer first wird einmalig beim Laden mit der Leit-
                        // adresse von Puffer belegt
int feld1[5]={7,3,9,2,6};  // extern: Der Vektor wird einmalig beim Laden belegt
int feld2[ ]={7,3,9};      // extern: Der Vektor wird einmalig beim Laden belegt
                        // Die Liste legt gleichzeitig die Elementezahl des Vektors fest
int feld3[50]={1,3};    // extern: Der Vektor wird einmalig beim Laden belegt
                        // Von den 50 Elementen werden aber nur 2 initialisiert
char z[ ]="Mein Konto"; // extern: Die Zeichenkettenvariable wird einmalig beim
                        // Laden belegt
int main(.......)
{ double A=5.5, y=6.3;  // automatic: Bei jedem Neustart initialisiert
  extern double f(double); // extern: Nennung einer erst später im Programm auftretenden
  .....                 // Funktion mit Ergebnistyp und Argumenttyp
  return 0;
}
double f(double x)      // extern: Funktionskopf
{ double A=7.6; b=8.5; y;  // automatic: A und b bei jedem Aufruf der Funktion initialisiert
  static double mat[3][2]=  // static: Einmalige Belegung der Matrix mat beim Laden
     {{11,12,13},{21,22,23}};
 double v[3]={1.0, 2.0, 3.0}; // automatic: Initialisierung bei jedem Aufruf der Funktion
  .....
}
```

Beispiel 23: Typumwandlungen

Daten treten uns in bestimmten Datentypen, z.B. int, double oder char, entgegen. Aus einer Int-Zahl kann man immer eine Double-Zahl machen. Umgekehrt kann das zu Fehlern führen. So lässt sich die Zahl **3.14** nicht ohne Datenverlust in eine ganze Zahl umwandeln. Auch die double Zahl 12345678909876543210 lässt sich nicht als ganze Zahl (Int-Zahl) speichern, da sie viel zu groß ist. Einige Eigenschaften der unterschiedlichen Datentypen:

char	8 Bit = 1 Byte	Zeichen mit den ganzzahligen Codierungen 0 – 255
short	16 Bit = 2 Byte	Ganze Zahlen von 0 bis 65535 bzw. -32768 bis 32767
int	32 Bit = 4 Byte	Ganze Zahlen von 0 bis etwa $4{,}3 \cdot 10^9$
float	32 Bit = 4 Byte	Kurze Gleitkommazahlen (Genauigkeit ca 5 signifikante Stellen)
double	64 Bit = 8 Byte	Normale Gleitkommazahlen (ca. 17 signifikante Stellen)

```
int i, j;   float a, b;     double x, y;   // Deklaration einiger Variablen
```

```
.....
j = i+1;                    // keine Typumwandlung, da alles int-Größen sind
a = i + 5. 0;               // Zuerst Umwandlung Wert von i in Double-Zahl, dann
                            // Addition der Konstanten (Typ double), dann Weg-
                            // lassen von Kommastellenlassen und speichern als float
```

Bei Funktionsaufrufen muss man manchmal einen bestimmten Wert erzwingen, wenn die Funktion bei einem Parameter einen ganz bestimmten Datentyp erwartet.

```
Z =  funk ( short ( 3 ) ,        float ( 3.14 ) ,        double ( 5 * i ) ) ;
        macht int zu short       macht double zu float   macht int zu double
```

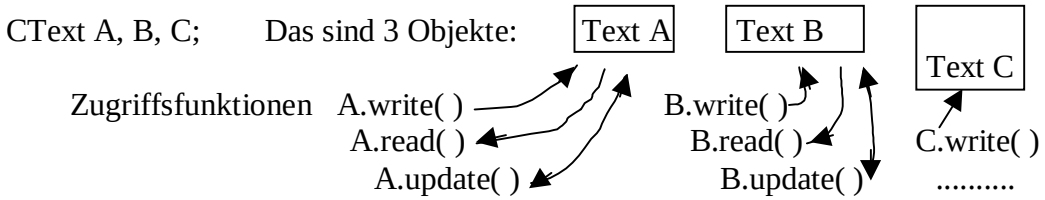
3. C++ mit MFC

3.1 Einführung in das Klassenkonzept

Eine **Klasse** ist eine Kombination aus Datenstruktur (Tabelle, Matrizen, Listen, ...)
und Zugriffsfunktionen zu den Daten (Lesen, Schreiben, Ändern, ...)

Die Datenstruktur wird gern vor dem Anwender verborgen, damit der Hersteller der Software diese ändern kann, ohne dass sich die Zugriffsfunktionen und ihre Wirkung ändert.

Beispiel 24: Die Klasse "Text"



Methode: Operation auf den Daten, Zugriffsfunktion (read, write, update, ...)

Botschaft (Massage): Aufruf einer Methode, d.h. Benutzung einer Zugriffsfunktion

MFC (Microsoft Foundations Classes): Die Firma Microsoft hat ca. 200 Klassen definiert, die alle von der Klasse *CObject* abgeleitet wurden. Damit **erben** alle Klassen die Eigenschaft der **Serialisierung** (d.h. die Daten, die die Klassen verwalten, lassen sich auf Datenträgern sichern).

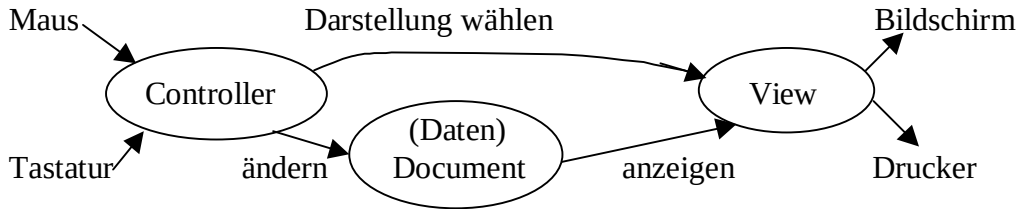
Wizard (Zauberer, Assistent): Assistentenprogramme helfen dem Programmierer, z.B. indem sie ein fertiges Programmgerüst bereitstellen, in das der Programmierer sein Problem einbaut. (Allein um ohne diese Hilfe ein Fenster auf dem Bildschirm zu öffnen, müssten wir Monate programmieren).

GUI (Graphical User Interface): Der Bildschirm läuft ausschließlich im Graphikmodus, d.h., alles was wir sehen, ist eigentlich ein Bild aus Pixeln aufgebaut. Zum GUI gehört aber auch, dass viel mit der Maus und wenig mit der Tastatur gearbeitet wird.

Document-View Architecture: Die Daten sehen anders aus, als das, was wir auf dem Bildschirm sehen. Beispiele:

- Eine Konstruktionszeichnung besteht aus Linien, Punkten, Kreisen, ... Eie Linie ist in den Daten definiert durch Anfangskoordinaten, Endkoordinaten, Strichdicke, Farbe, ..., aber auf dem Bildschirm ist es eine Ansammlung gefärbter Pixel.
- Ein Satz in einem Worddokument besteht aus den Buchstaben und Sonderzeichen. Hinzu kommen jedoch Angaben zur Schriftart, Schriftgröße, Farbe, Unterstreichung, ... Was wir sehen, ist das erzeugte Bild unseres Satzes - einfach gefärbte Pixel.

Ein Programm in dieser Architektur hat 3 Hauptbestandteile:



Das Controller-Programm registriert jeden Tastendruck (auch das Loslassen), jede Mausbewegung und jede Maustaste. Wird eine Schaltfläche angeklickt, sendet es Nachrichten, entweder an das Document-Program oder an das View-Program.
Das Document-Program verwaltet hauptsächlich die Daten (Lesen, Schreiben, Ändern, ...)
Das View-Program holt sich die Daten und bereitet sie graphisch auf, um eine Ansicht des Objekts auf den Bildschirm oder auf den Drucker zu bringen.

Anwendungsgerüst heißt eine bestimmte Programmkombination aus Controller-, Document-, Fensterverwaltung und View-Program zusammen mit einer bestimmten *Dokumentenvorlage*. Ein Wizard (Assistentenprogramm) stellt das Anwendungsgerüst nach der angeklickten Vorlage zusammen. Wir verwenden im Praktikum ausschließlich das **SDI** (Single Document Interface).

- Klasse CDocument verwaltet die Daten (Text, Zahlen, Bilder, ...)
- Klasse CView stellt die Daten auf unterschiedliche Weise dar (Tabelle, Text, Graphik, ..)
- Klasse CFrameWnd liefert uns das Fenster mit Titel, Menüleiste, Scrollbalken, ...
- Klasse CDocTemplate (Dokumentenvorlage) lädt die 3 vorangehenden Objekte, bleibt selbst aber verborgen.

Ungarische Notation heißt die Namensgebung von Variablen und Programmen, wenn man mit dem Namen, der eigentlich frei wählbar ist, eine Information an den Nutzer übermittelt. Sie können eine Variable natürlich *Rumpelstilzchen* nennen, aber wenn es sich um eine Quartalssumme handelt, dann findet man leicht einen besseren Namen, z.B. *Quartalssumme*.

Namen der Form	**Cxyz**	sollten **Klassen** benennen
Namen der Form	**pXyz**	sollten **Pointer** benennen
Namen der Form	**m_Xyz**	sollten **Membervariablen von Klassen** benennen
Namen der Form	**n_Xyz**	sollten **Integerzähler** benennen
Namen der Form	**lpszXyz**	sollten **Pointer auf Zeichenketten alten Typs** benennen (Zeichenketten, die mit ∅ enden, eigentlich die Abkürzung von "long pointer to strings zero-terminated")
Namen der Form	**OnUpdate( )**	sollten **Funktionen** benennen (jede Silbe fängt groß an)
Namen der Form	**WM_UPDATE**	sollten **Nachrichten** benennen (eigentlich sind es Konstantennamen, da Nachrichten einfach Zahlen sind)

Nachrichten lösen Aktionen in den einzelnen Programmteilen aus. Immer, wenn ein Programmteil seine Arbeit beendet hat, oder es so programmiert ist, dass ein Mausklick, eine Taste oder ein Zeitsignal es unterbrechen können, dann wird der Nachrichtenblock durchgesehen, ob eine noch nicht bearbeitete Nachricht vorliegt. Wenn ja, dann wird die zugehörige Funktion aufgerufen. Z.B. veranlasst ein Klick auf Menüpunkt "Datei neu" die Nachricht ON_COMMAND_FILE_NEW, und diese bewirkt dann den Aufruf der Funktion OnFileNew( ).

Nachrichten sind als Zahl codiert. Jede Zahl hat ihre Konstantenbezeichnung, z.B. WM_UPDATE. Zu manchen Nachrichten gehört noch Zusatzinformation, z.B. eine Fenstergröße SIZE (meist in Pixeln gemessen).

Kommunikation zwischen Programmteilen: Außer dem Absetzen von Nachrichten gibt es noch den Aufruf bestimmter Funktionen:

- Der Aufruf **UpdateAllViews** in **CDocument** erneuert alle Views
- Der Aufruf **GetDocument** in **CView** liefert CView den Zugriff auf die Daten

3.2 Aufbau einer Klasse, Begriffsbestimmungen

Eine Klasse kann eigenständig auftreten wie ein Funktionsprogramm etwa, oder aber am Beginn eines Blockes definiert werden in der Form **{class Cxyz ..... }**

Falls eine Klasse, z.B. CName, eigenständig definiert wird, dann existieren meist 2 Files:

1. Die Headerdatei **Name.h** zur Klasse CName
2. Die Pogrammdatei **Name.cpp**, falls die Klasse externe Funktionen verwendet

Inline heißt eine Funktion, wenn ihr gesamter Programmtext an jeder Stelle, an der die Funktion arbeiten soll, vollständig in den Quelltext des C-Programms eingefügt wird. Das macht man nur mit Funktionen, die einen kurzen Programmtext haben.

Extern heißt eine Funktion, deren Programmtext nur einmal existiert. An den Stellen im C-Programm, an denen die Funktion arbeiten soll, wird ein Funktionsaufruf eingesetzt. Da Funktionsaufrufe meist nur wenige Zeichen lang sind, spart man immens an Speicherplatz und Übersetzerzeit, insbesondere dann, wenn der Programmtext der Funktion selbst sehr lang ist.

pDC heißt der Pointer auf den Device Context. Das ist eine Sammlung von Funktionen zur Arbeit mit dem Viewfenster. Z.B. gestattet die Funktion TextOut(....) aus dem Device Context Texte im Viewfenster zu platzieren.

pDoc heißt der Pointer zum Datenverzeichnis des Dokuments. Man braucht ihn, um vom View-Program auf die Daten zugreifen zu können.

–> heißt **Entreferenzierungspfeil**. Links vom Pfeil steht zumeist ein Pointer (Adresse eines Verzeichnisses), rechts vom Pfeil wird ein Element des Verzeichnisses genannt. Auf dieses Element wird zugegriffen. Z.B. wählt pDC -> TextOut(....) die Funktion TextOut aus.
Der Entreferenzierungspfeil wirkt wie (*pDC).TextOut(....)

:: ist der **Scope Operator** (Bereichs-Operator). Vor dem zweifachen Doppelpunkt steht der Name der Klasse, dahinter ein Element dieser Klasse, z.B. eine Funktion.

Schutzstufen: Für die Daten und Funktionen einer Klasse gibt es 3 Schutzstufen:

private:	Daten oder Funktion nur in der Klasse selbst verfügbar
protected:	Auch in abgeleiteten Klassen (Subklassen) verfügbar
public:	Für jeden verfügbar, insbesondere für die Anwender der Klasse

Membervariable sind schützenswerte oder für den Anwender wichtige Variablen einer Klasse
Memberfunktionen sind die (meist public) Zugriffsmethoden und Funktionen einer Klasse
Konstruktoren richten neue Objekte ein (Speicherplatz aquirieren, Initialisierung usw.)
Destruktoren bauen Objekte zurück (Speicher freigeben, Adressverweise löschen usw.)
Nutzer sind Personen, die eine Klasse benutzen. Sie kennen nur das Interface, haben nur Zugriff auf public Daten und public Funktionen.
Implementoren sind Personen, die eine Klasse erfinden und programmieren. Sie haben den Quelltext der Klasse, und damit Zugriff auf alle Details der Daten und ihrer Bearbeitung.

3.3 Die Klasse CNurEineZahl (Beispiel 25)

Das File **NurEineZahl.h**

```
class CNurEineZahl                  // Eine Klassendefinition beginnt mit class
{ int m_i;                          // protected Membervariable
  public:                           // ab hier wird alles public
    CNurEineZahl( ) { m_i = 0; }    // Inline-Konstruktor mit Initialisierung
    virtual  ~CNurEineZahl( );      // Destruktor als externe Funktion
   void Set (int x ) { m_i = x; }   // Inline-Zugriffsfunktion "Wert setzen"
   int Show( )  { return( m_i ; }   // Inline-Zugriffsfunktion "Wert zeigen"
   double Fakul ( );                // Externe Zugriffsfunktion "Fakultät"
};                                  // Wichtig!! Das Semikolon nach der Klammer
```

Das File **NurEineZahl.cpp** mit den beiden externen Funktionen

```
#include <NurEineZahl.h>            // Macht den Klassenheader bekannt
CNurEineZahl :: ~CNurEineZahl( )    // Kopfzeile der Destruktorfunktion
                          {    }    // Der lächerlich kleine Programmtext dazu
double CNurEineZahl :: Fakul ( )    // Kopfzeile der Fakul-Funktion
{ double F;                         // lokale Variable der Funktion
   int i, imax, MAX=100;            // weitere lokale Variable, eine initialisiert
   F=1.0;                           // Startwert für die Fakultätsberechnung
   if ( m_i <= MAX ) { imax = m_i ; }   // F=x! wird nur bis x=100 ausgeführt. Ist x>100,
                 else { imax = MAX; }   // dann wird einfach F=100! zurückgegeben
   for( i=1; i<=imax; i++) { F=F*i ;}   // z.B. 5! = 1·2·3·4·5
   return (F);                      // Wertrückgabe und Rückkehr zur Aufrufstelle
}                                   // Funktions- und gleichzeitig hier auch Fileende
```

Wir wollen unsere selbstgebastelte Klasse CNurEineZahl jetzt in einem Programm namens *Beispiel* verwenden, und zwar in der Funktion OnDraw( ), die das View-Fenster einer SDI-

Anwendung gestaltet, also genau das Ausgabefenster unseres Programms *Beispiel*, das wir auf dem Bildschirm sehen.

```
#include <NurEineZahl.h>                    // Unsere Klasse hier bekannt machen

// Es folgen 4 Standardzeilen, die der Assistent herstellt. Wir sollten sie nicht ändern
void CBeispielView :: OnDraw(CDC *pDC)      // Kopfzeile der OnDraw-Funktion
{ CBeispielDoc *pDoc=GetDocument( );        // Macht die Dokumentdaten verfügbar
  ASSERT_VALID(pDoc);                       // Eine Überprüfung auf Vollständigkeit
  // Zur Erledigung: Hier Code einfügen        Freundlicher Hinweis des Assistenten

  CNurEineZahl  A, gamma;                   // Deklaration zweier Objekte
  int k;                                    // neue lokale Variable in OnDraw( )
  double DF;                                // neue lokale Variable in OnDraw( )
  char zeile[100];                          // neue lokale Zeichenkettenvariable

  A.Set(5);                                 // m_i  von Objekt  A  wird 5 gesetzt
  gamma.Set(3);                             // m_i  von Objekt  gamma wird 3 gesetzt

  k = A.Show( ) + gamma.Show( ) + 100;      // Kleine Berechnung: Ergibt k=108
  sprintf( zeile, "Summe=%5i", k );         // Herstellung einer Textzeile auf zeile
  pDC->TextOut(100,120,zeile);              // Ausgabe ins Fenster bei x=100, y=120

  DF = A.Fakul( ) – gamma.Fakul( );         // Wie DF = 5! – 3! = 114.0
  sprintf( zeile, "Fakul-Differenz=%10.1f", DF );  // Herstellung einer Textzeile auf zeile
  pDC->TextOut(100,150,zeile);              // Ausgabe ins Fenster bei x=100, y=150
}                                           // Ende von OnDraw in BeispielView.cpp
```

Das Pixel x=0, y=0 ist das oberste linke Pixel in unserem Viewfenster. Bei wachsendem x gehen wir nach rechts, bei wachsendem y nach unten. Die Funktion sprintf(zielkette, format, ausgabeliste) stellt auf der Zielkette aus Zahlen und Text eine ausgebbare Zeichenkette zusammen, die wir anschließend ins Fenster setzen können. Will man nur Text ausgeben, kann man ihn direkt in der Funktion TextOut( ) als Literal angeben, z.B.

```
pDC->TextOut(100, 70, "Ergebnisse der Beispieldaten:");
```

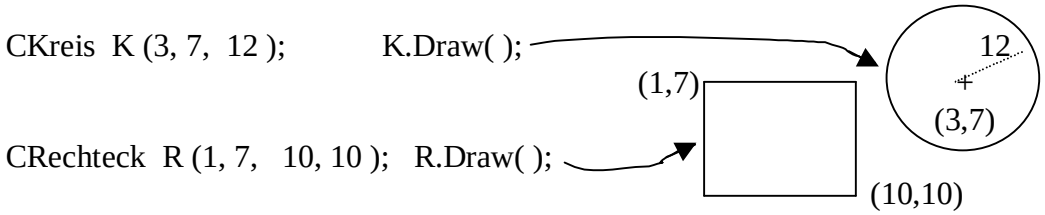
Übung 11:	Wir definieren eine Klasse CNurEinText mit den 2 Inline-Funktionen Put( ) und Get( ) und wenden die Klasse in einem Beispiel an. Put( ) soll einen Text abspeichern, Get( ) ihn zeigen. Für die Zeichenkettenverarbeitung benutzen wir jedoch Zeichenketten, die mit der Klasse CString deklariert werden (neue Form der Zeichenketten).
	
	Ein unvollständiger Schnellkursus für die Klasse CString:
	
	CString a, b, c, Xyz; // Deklaration von 4 Zeichenkettenvariablen
	a = "100 Jahre Einsamkeit"; // Zuweisung eines Literals
	b = a; // Zuweisung zu einer anderen Zeichenkette
	c = a + b; // Hängt Kette a und b zusammen
	a.MakeUpper( ); // macht "100 JAHRE EINSAMKEIT" daraus
	a.MakeReverse( ); // macht "TIEKMASNIE ERHAJ 100" daraus

3.4 Basiswissen C++ (mehr informativ)

In C++ dürfen überall neue Variable deklariert werden. In C nur am Blockanfang.

Vererbung: Eine neue, abgeleitete Klasse erbt alles von der Basisklasse (Datenstruktur, Zugriffsfunktionen). Eine Ausnahme bilden der Konstruktor und der Destruktor. Die müssen immer neu definiert werden.

Polymorphie: Man definiert eine Basisklasse, z.B. CFigur. Daraus leitet man Subklassen ab, z.B. Kreis, Rechteck, ... Die Funktionsnamen der Zugriffsfunktionen, z.B. Set( ), Draw( ), Move( ), ..., stimmen in allen Subklassen überein, machen aber etwas Unterschiedliches.



virtual: Eine **virtual Function** wird in der Basisklasse als Platzhalter ohne jeden Inhalt definiert. In den Subklassen kann der Inhalt dann unterschiedlich definiert werden. Auf diese Weise entsteht die Polymorphie.

Function Overloading: Dieselbe Funktion akzeptiert unterschiedliche Argumenttypen, manchmal sogar unterschiedliche Argumentanzahl. Beispiel: Wir rufen die Wurzel auf:

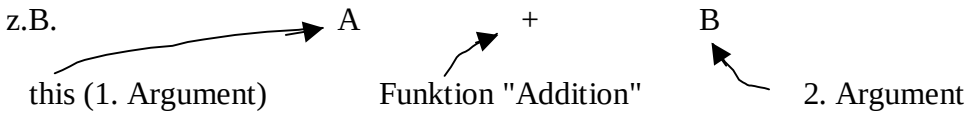
W = sqrt(7.3);	Aufruf O.K. Die Wurzel erwartet nämlich ein double-Argument.
W = sqrt(7);	Aufruf **nicht** O.K. Die Wurzel bekommt hier ein int-Argument.

Um solche Fallen zu minimieren, programmiert man oft dieselbe Funktion für jeden Argumenttyp neu, hier also Wurzel aus double-Argument, Wurzel aus float-Argument, Wurzel aus int-Argument usw. Der Compiler kann die Funktionen trotz ihres gleichen Namens (alle Wurzelfunktionen heißen sqrt) am Argumenttyp auseinanderhalten.

Operator-Overloading: Man missbraucht die bereits vorhandenen Operatoren + – * / usw. für ganz andere Operationen mit neuen Datentypen, z.B. Verketten von Strings. Natürlich steckt da erhebliche Programmierarbeit von Profis dahinter. Wir begnügen uns mit dem Wissen, dass es geht.

```
CString   a("Adam"), b("Eva"), c;          // Deklaration von 3 Stringvariablen, 2 initialisiert

          c = a + "und" + b;               // ergibt c = "Adam und Eva"
```

return(*this); Ist eine Konstruktion bei Funktionen, die als Ergebnis nicht einfach Zahlenwerte zurückliefern, sondern berechnete oder veränderte Objekte. Die Anweisung return(*this); gibt als Funktionsergebnis die Adresse des ersten Arguments zurück (falls die Argumente Objekte sind). Bei simplen numerischen Funktionen, wie sqrt(x) braucht man diese Konstruktion nicht, wohl aber beim Operator Overloading, da hier die Operationen von Funktionen ausgeführt werden.



Identitäten: Es gibt keine zwei identischen Äpfel. Ein Objekt ist nur mit sich selbst wirklich identisch. Die Zeigerkopie verwirklicht diese Erkenntnis:

Ändert sich das Objekt, dann ändert es sich für den kopierten Zeiger auch.

Fehlerbehandlung mit try, throw, catch: Statt im Vorfeld einer Berechnung mühsam zu recherchieren, ob denn die Berechnung mit den vorliegenden Daten auch ja nur gut gehen könnte, tappt man beim **try, throw, catch** einfach in alle Fehlerfallen und stellt hinterher fest, was schiefgelaufen ist. Der Effekt ist derselbe, aber mit weniger Aufwand.

Beispiel: Der unbeugsame Programmierer testet vor jeder Division, ob der Nenner Null ist, und bricht in einem solchen Fall das Programm mit einem gezielten Fehlermeldung ab. Der Try-Throw-Catch-Programmierer macht einen Versuch (try), die Division "wirft" einen Fehler, falls sie durch die ungeliebte Null teilen soll (throw), und der catch-Teil fängt den Fehler auf und macht die Meldung. Besonders bei Funktionen, die andere programmiert haben, verlässt man sich gern auf das "Fehlerwerfen", das sind insbesondere die vielen Funktionen der Ein- und Ausgabe von Daten, wie ReadString( ), WriteString( ), usw. Natürlich sind so allgemeine Fehlermeldungen, wie "wenden Sie sich an den Hersteller" nicht wirklich hilfreich.

4. Einführung in Java

Java wurde ab 1990 von der Sun-Corporation aus C++ entwickelt. BlueJ ist eine Entwicklungsumgebung für Java-Programme der Monash University, Melbourne, Australia.

4.1 Grundlagen von Java

Primitive Datentypen sind byte, short, int, long (64-Bit-Integer), float, double, char (hier in Java aber 16-Bit Unicodezeichen), boolean (logisch true/false).

Objekttypen: Die Variable ist ein Zeiger, der auf das Objekt zeigt. Bei der einfachen Zuweisung **A=B;** wird lediglich der Zeiger B nach A umkopiert, so dass anschließend beide Zeiger auf dasselbe Objekt zeigen.

Felder, z.B. Vektoren und Matrizen, sind solche Objekttypen.

Wrapper-Klassen: Als Objekt verpackter primitiver Typ. Man schreibt **B**yte statt **b**yte, **S**hort statt **s**hort, usw.

Anweisungen if, while, do-while, for bleiben so, wie in C++.

switch-Anweisung: ist eine Mehrfachverzweigung (die es auch in C und C++ gibt). Der allgemeine Aufbau einer switch-Anweisung und ein Beispiel dazu (fehlt break, wird der nächste Fall gleich mitgenommen):

switch (Integer-Ausdruck) { case 1: Anweisung oder Block; break; case 2: Anweisung oder Block; break; ..... case n: Anweisung oder Block; break; default: Anweisung oder Block; }	**switch(k)** **{case (-1): a=5. 5; break;** **case 0: { a=6. 6; b= –1. 0;} break;** **case 1:** **case 2: { a=7. 3; b=+1. 0;]; break** **default: { a=1. 0; b=0. 0;}** **}**

Hat k den Wert -1, dann wird a=5.5 gesetzt und die Switch-Anweisung verlassen.
Hat k den Wert 0, wird a=6.6 und b=-1 gesetzt und die Switch-Anweisung verlassen.
Hat k den Wert 1 oder 2, wird a=7.3 und b=1 gesetzt und die Switch-Anweisung verlassen.
Hat k keinen dieser Werte, wird a=1 und b=0 gesetzt.
Endet eine Anweisung mit **return**, dann kein **break** (Es kann nicht erreicht werden)

Konstanten in Java

final double K=10.7 // definiert eine Konstante K mit Wert 10,7
final int NUM=88; // definiert eine Konstante NUM mit Wert 88

Felder (Vektoren und Matrizen) in Java

int [ ] X; // Deklaration eines Vektors X, d.h. einer Feldvariablen
double [ ][ ] mat; // Deklaration einer Matrix mat, d.h. einer Feldvariablen.

// Nur deklarierte Feldvariablen haben noch keinen Speicherplatz reserviert. Der wird erst aquiriert, wenn ein neues Objekt dieser Klasse angelegt wird:

ma t = n e w d o u b l e [1 0 0] [1 0] ; // (100, 10)-Matrix neu anlegen
Zeilenzahl Spaltenzahl

mat [3][7] = K ; // Ein Matrixelement wird belegt

X = new int [50]; // Speicherplatz für Vektor X wird angefordert

for (i=0; i<50; i++) X[i] = i*i+1; // Alle 50 Elemente von X werden mit den Werten
// i^2+1 belegt, d.h. X_0=1, X_1=2, X_2=5, X_3=10, ...

Bibliotheken in Java, z.B. Klasse Math

i=Math.abs(j); // Integerbetrag, wirkt wie i = | j |
z = Math.abs(x); // Gleitkommabetrag, wirkt wie z = | x |
i = Math.round((float) x); // rundet x auf die nächste ganze Zahl. Da die Funktion
// round ein float-Argument erwartet, wird ein **cast**

```
                                    // benutzt, d.h. eine Zwangstypumwandlung
z = Math.log(x);                    // Natürlicher Logarithmus  z = ln(x)
z = Math.exp(x);                    // Exponentialfunktion  z=e^x
```
Weitere Funktionen sind z.B. Math.sin( ), Math.cos ( ), Math.sqrt( )
```
z = Math.pow(x, y);                 // Powerfunktion  z = x^y  (nur für x>0)
z = Math.min( i, j );               // Minimum von 2 Integer oder von 2 Double-Werten
z = Math.max( x,  3.7);             // Maximum von 2 Integer oder von 2 Double-Werten
z = Math.random( );                 // Zufallszahl   0 ≤ z < 1
```

cast

Casts sind erzwungene Typumwandlungen eines Wertes, z.B. von double in float

i = Math.round (**(float)** x);

4.2 Beispiel Ticketautomat (Beispiel 26)

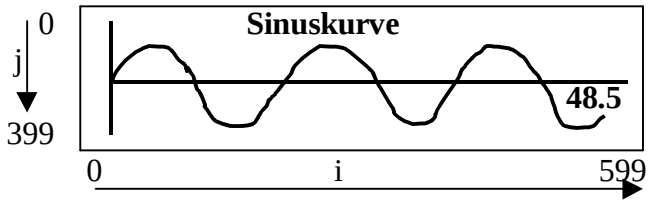
Ein Ticketautomat nimmt Geld und gibt ein Ticket aus, wenn genug eingezahlt wurde. Ein Tageszähler bildet die Einnahmensumme für den Tag.

```
public class Ticketautomat                  // Eine öffentlich verfügbare Klasse
{ private int preis;                        // Deklaration von Membervariablen
  private int bishergezahlt;
  private int tagessumme;
  public Ticketautomat ( int tp )           // Kopfzeile des Konstruktors
        {  preis = tp ;                     // Inline-Codezeilen des Konstruktors
           bishergezahlt = 0 ;
           tagessumme = 0 ;
         }
public int gibPreis( )  { return (preis ); }        // 1. Zugriffsmethode Inline-Code
public int gibBisherGezahlt ( )             // Kopfzeile 2. Zugriffsmethode Inline
        { return ( bishergezahlt); }
public void geldEinwerfen ( int betrag )    // Kopfzeile 3. Zugriffsmethode Inline
      { bishergezahlt = bishergezahlt+betrag; }
public void ticketDrucken ( )               // Kopfzeile 4. Zugriffsmethode
  { System.out.println("ABBA-Line");        // Inline Code-Zeilen für das Drucken
    System.out.println("Preis="+preis+" Cent");
    System.out.println( );                  // leere Zeile
    tagessumme += bishergezahlt;            // Tagessumme akkumulieren
    bishergezahlt = 0;                      // Null setzen fürs nächste Ticket
  }
}                                           // Ende Klassendeklaration
```

Übung 12:	Unser Ticketautomat soll 2 Linien kennen (Linie 1, Linie 2). Wir haben also als Daten einen *preis1*, einen *preis2* und *linie*. Benutzen Sie in der Funktion gibPreis den Parameter *lnum* wie Liniennummer und eine switch-Anweisung im Inline-Code. Zusätzlich brauchen wir die Zugriffsfunktion linieWaehlen, ebenfalls mit Argument *lnum*. Auch beim Ausdrucken verwenden Sie eine Switch-Anweisung. Im default-Fall hat der Kunde keine Linie gewählt. Sagen Sie ihm das.

4.3 Beispiel Sinuskurve (Beispiel 27)

Für die Ausgabe einer Graphik benutzen wir eine Klasse "Leinwand", die ich auf einer Diskette mit ins Praktikum bringe. Die Anwendung entnehmen Sie dem Beispiel:

Das Rechteck zeigt die Graphik, die wir erzeugen wollen. Die Indizes i und j zählen die Spalten- und die Zeilenpixel. Überschrift ist "Sinuskurve", Endwert der x-Achse sei 48,5	

```
public class Sinuskurve
{ private double frequenz;
  public Sinuskurve ( double F )                    // Kopfzeile Inline-Konstruktor
        { Frequenz = F; }
public void setzeFrequenz( double F )               // Kopfzeile 1. Zugriffsmethode
        { Frequenz = F; }
public double  gibFrequenz ( ) { return (Frequenz); }  // 2. Zugriffsmethode inline

public void zeichneGraphik ( )                      // Kopfzeile 3. Zugriffsmethode
{ // Erzeuge Fenster "Leinwand" mit 600*400 Pixeln und einem Titel im Fensterrahmen
  Leinwand   leinwand = Leinwand.gibLeinwand ( "Beispiel Sinuskurve", 600, 400 );
  // leinwand.loeschen( );                          // Mache die Leinwand weiß

  int i, j, ia, ie;                                 // Lokale Zähler (Pixelnummern)
  double x, y;                                      // Punktkoordinaten der Kurve

  for ( i=0; i<500; i++)                            // 500 Pixel in x-Richtung
  { x = Frequenz * i ;                              // Berechne Argument in Radiant

   // Da wir beim Zeichnen eines Kurvenstücks die Funktion
   // leinwand.linie(farbe,  x1,y1, x2,y2) benutzen mit Startpixelkoordinaten des Linienstücks
   // (x1,y1) und Endpixelkoordinaten (x2,y2), setzen bei i=0 wir einen Startpunkt..
   // Ist aber i>0, dann ziehen wir ein winzig kleines Linienstück vom alten Punkt zum neuen
   // Punkt

   if ( i == 0 ) { ia=0, ja=0; j=0; }               // Startpunkt bei i=0 (alter Punkt)
   else
   { y = Math.sin(x) * 150;                         // Maximum wird 150 Pixel hoch
     j = Math.round((float) y);                     // Runden auf ganze Pixelanzahl

     // Wir zeichnen das Linienstück, müssen aber daran denken, dass die Nulllinie unserer
     // Sinuskurve 200 Pixel vom oberen Rand entfernt ist, und unsere y-Achse 10 Pixel vom
     // linken Rand eingerückt ist

     leinwand.linie("schwarz", ia+10, 200–ja, i+10, 200–j );      // Linienstück zeichnen

     // Wir machen den neuen Punkt (i, j) zum alten Punkt (ia, ja)
```

```
    ia = i;     ja = j;
  }                                                    // Ende des else-Zweigs
}                                                      // Ende der for-Schleife
```

```
// Zum Abschluss platzieren wir den Titel "Sinuskurve" (Koordinate ist linkes unters Pixel
// des ersten Buchstaben des Textes)

leinwand.text ( "rot", 200,20, "Sinuskurve" );          // Titel

// Wir zeichnen die x- und die y-Achse als Striche

leinwand.linie( "schwarz", 10, 10,      10, 390 );      // y-Achse von 10,10 nach 10,390
leinwand.linie( "schwarz", 10, 200,   510, 200 );      // x-Achse

// Um eine Zahlenausgabe zu zeigen, schreiben wir den Endwert von x unter das Ende der
// x-Achse. Dabei kürzen wir aber die Zahl der Dezimalstellen so, dass die ausgegebene
// Zahl aus maximal 5 Zeichen besteht bzw. weniger (das Minimum aus 5 und der
// tatsächlichen Zahlenlänge)

string Endwert=string.valueOf(x);                      // Konvertiere Zahl in Zeichenkette
leinwand.text("blau", 480, 230, Endwert.substring(0, Math.min(5, Endwert.length())));

}  // Ende der Funktion zeichneGraphik

} // Ende der Klasse
```

4.4 Numerische Lösung von gewöhnlichen Differerenzialgleichungs-systemen mit dem Euler-Verfahren (Beispiel 28)

Eine Abkühlkurve nach dem Wärmeleitungsmodell folgt der gewöhnlichen DGL mit konstanten Koeffizienten:

$$\underset{[s]}{T} \cdot \underset{[K/s]}{\dot{x}(t)} + \underset{[K]}{x(t)} = \underset{[K]}{x_B}$$

Die Abkühlung pro Zeiteinheit in [K/s] ist bei reiner Wärmeleitung proportional zur Temperaturdifferenz x_B–x(t). Der Proportionalitätsfaktor ist 1/T. Die Zeitkonstante der Abkühlung sei T=25 [s], der Beharrungswert, gegen den die Temperatur strebt, sei 20 [°C]. Die Starttemperatur sei x_0=90 [°C].

Die algebraische Lösung dieser einfachen DGL ist bekannt. Es gibt zwei gleichwertige und ineinander überführbare Lösungen:

(1) $x(t) = x_B + (x_0 - x_B) e^{-t/T}$

(2) $x(t) = x_0 + (x_B - x_0) (1 - e^{-t/T})$

Die numerische Lösung, z.B. nach dem Euler-Verfahren, liefert keine Formel, sondern aufeinanderfolgende t- und x-Wertepaare, die graphisch dargestellt die Lösungskurve bilden.

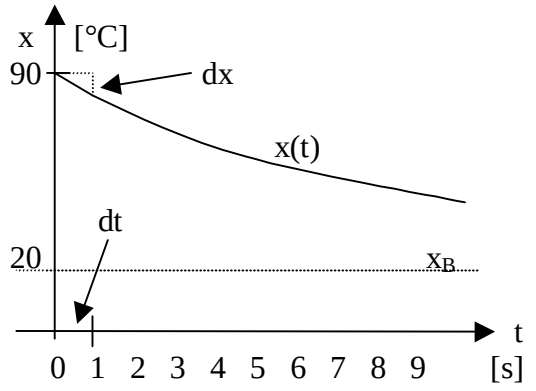
Das Euler-Verfahren: Wir formen die DGL um in $\dot{x} = (x_B - x)/T$ oder dx/dt=(x_B–x)/T oder dx=((x_B–x)/T) dt . Für einen Zeitschritt wählen wir dt=1 [s]. Wir starten mit x = x_0 =90 °C und erhalten

(1) dx = ((20–90)/25)·1 = –2,80 [K].

Wenn wir in der Zeit um einen Schritt dt=1 fortschreiten, sinkt die Temperatur um dx = –2,80 Grad. Der neue x-Wert ist dann

(2) $x_{neu} = x_{alt} + dx = 90 - 2{,}80 = 87{,}2$ °C

Jetzt setzen wir x_{neu} statt 90 in Gl (1) ein und errechnen das nächste dx und das nächste x_{neu} usw.



Die Umwandlung einer DGL oder eines Systems gekoppelter DGLs als Aufbereitung für eine numerische Lösung folgt demnach einfachen Regeln, die an zwei Beispielen gezeigt werden:

(1) $T\,\dot{x} + x = x_B \quad \Rightarrow \quad \begin{cases} \dot{x} = \dfrac{x_B - x}{T} & \textit{umgeformte DGL} \\ x = x + \dot{x}\,dt & \textit{Euler – Integration} \end{cases}$

(2) $x^2\,y'' - y'^2 + y''\,y = \dfrac{x}{1+z^2}$ und $T\,z' + z\,y = K$

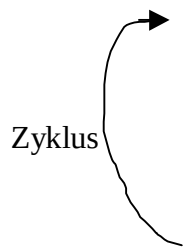
mit den Konstanten T und K, und den gesuchten Lösungen y(x) und z(x). Das Gleichungssystem (2) ist ein System von zwei gekoppelten nichtlinearen DGL. x ist die unabhängige Variable.

Als erste Umformung löst man beide DGLs des Systems (2) jeweils nach ihrer höchsten Ableitung auf:

$$y'' = \frac{\dfrac{x}{1+z^2} + y'^2}{y + x^2} \qquad \text{und} \qquad z' = \frac{K - z\,y}{T}$$

Dann muss man Startvorgaben machen, d.h. Anfangswerte und Rechenschrittweite vorgeben: x=0, y=y_0, y'=y'_0, z=z_0, dx=0,001 (optimalen dx-Wert findet man durch probieren).

Die Berechnung der Lösungswerte ist ein iterativer Prozess, bei dem immer wieder dieselben Gleichungen durchlaufen werden. Man rechnet, bis ein vorgegebener x-Wert erreicht ist.



$y'' = \dfrac{\dfrac{x}{1+z^2} + y'^2}{y + x^2}$	(1)	y" aus umgeformter DGL 1 berechnen
y ' = y ' + y" dx	(2)	y ' (x) nach der Euler-Integration
y = y + y ' dx	(3)	y(x) nach der Eulerintegration
z ' = (K – z y ') / T	(4)	z ' aus umgeformter DGL 2 berechnen
z = z + z ' dx	(5)	z(x) nach der Euler-Integration
x = x +dx	(6)	Erhöhe das x um einen Schritt

```
public class Abkuehlkurve
{ private double x0, xB, T, te;                          // Objektdaten (Kurvenparameter)
  public Abkuehlkurve( double xa, double xe, double Tk, double tend)     // Konstruktor
          { x0 = xa ;  xB = xe ;  T = Tk ; te = tend; }       // Inline-Code des Konstruktors

   public void zeichneAk ( )                             // Kopfzeile Zugriffsfunktion
  {  // Richte ein Graphikfenster "Leinwand" mit Titel "Abkühlkurve" ein
     Leinwand  leinwand = Leinwand.gibLeinwand ( "Abkühlkurve", 600, 400 );
     leinwand.loeschen( );                               // Mache den Fensterinhalt weiß

      // Wir verteilen 500 Ausgabepunkte (500 Pixel) der Kurve auf das Zeitintervall 0 bis te.
     // Wir definieren deshalb das Ausgabezeitintervall   dtaus = te/500.  (Endzeit / 500 )
     // Immer, wenn ein Stück Kurve integriert ist und ein Ausgabezeitschritt  dtaus  um ist,
     // erhöhen wir unseren Pixelzähler i um 1 und zeichnen ein Stück der Kurve

     int  i, j, ialt, jalt;                              / Pixelkoordinaten neu, alt
     double  x = x0, t = 0. 0,  dt = 0. 001;             // Startwerte, Rechenschrittweite
     double  xp,  dtaus = te/500;                        // lokale Hilfsvariable
     final massstab = 3. 0;                              // Zeichenmaßstab (Konstante)

     // Aus der Starttemperatur x0 und dem Maßstab lässt sich das Startpixel berechnen
     // Beachten müssen wir jedoch, dass die t-Achse 350 Pixel vom oberen Fensterrand,
     // die senkrechte x-Achse 10 Pixel vom linken Fensterrand
     // entfernt ist, und dass j kleiner wird, je größer der Ordinatenwert des Punktes ist

     ialt=10;  jalt= 350 - Math.round( (float) (x*massstab) );       // Startpixel der Kurve

     do                                                  // Beginn einer do-while-Schleife
     { xp = ( xB - x ) / T;                              // Anstieg im Punkt t laut DGL
       x = x + xp*dt;                                    // Eulerintegration der Kurve x(t)
       t = t+ dt;                                        // Zeit erhöhen

        // Wir testen, ob schon ein Druckzeitschritt   dtaus  um ist (Vorsicht,  i startet bei 10)

       if ( t > ((ialt + 1 – 10 )*dtaus) )
       { // Im Ja-Fall berechnen wir die neuen Pixelkoordinaten und zeichnen ein Stück Kurve
         i=ialt+1;                                       // neue Pixelnummer t-Richtung
         j = 350 – Math.round( (float) ( x*massstab));   // neue Pixelnummer x-Richtung
         leinwand.linie("schwarz", ialt, jalt,   i, j );  // Linienstück zeichnen
         ialt=i;  jalt=j;                                // mache neuen zum alten Punkt
       }                                                 // Ende der if-Anweisung
     }while (t < te );                                   // Ende der do-while-Schleife

     // Wir wollen jetzt noch die beiden Achsen zeichnen
     leinwand.linie("schwarz", 10, 360,  10,10);         // senkrechte x-Achse
     leinwand.linie("schwarz",  5, 350,   520, 350 );    // waagrechte t-Achse
  }                                                      // Ende der Zeichne-Funktion
}                                                        // Ende der Klasse
```

Computeranwendungen - vom Experiment zum Dokument

Vorlesung mit Praktikum

Prof. Dr. Stefan von Weber, HS Furtwangen, FB MuV

Vorlesung	Lecture
1. Einführung in LabView ® Sensoren im AT-Praktikum, Messcomputer, Datenlogger	1. Introduction to LabView ® Sensors in the AT practical course, measuring PC, data logger
2. Relationale Datenbanken, Access ®	2. Relational data retrieval systems, Access ®
3. Einführung in EXCEL ® WinStat Statistiksoftware	3. Introduction to EXCEL ® WinStat statistical software
4. Kurze Einführung in MS-WORD ®	4. Short introduction to MS-WORD ®
Praktikum	Practical course
1. Besuch im AT-Labor: LabView, Messcomputer, Datenlogger	1 Visiting the AT-Lab: LabView, measuring PC, data logger
2. Acces Datenbank *Gruen*	2. Access data collection *Gruen*
3. Statistik und Stabwerkberechnung mit EXCEL (2 Termine) Statistik mit WinSTAT	3. Statistics and computing a set of linear members with EXCEL (2 visits) Statistics with WinSTAT

Literatur / References

1. Matthiessen & Unterstein: *Access,* Addison-Wesley

2. Said Baloui: *EXCEL*, Makt & Technik

1. Einführung in LabView ®

LabView wurde ab 1983 von Jeff Kodosky von der Firma National Instruments entwickelt. Eingesetzt wird LabView für Messen, Steuern, Regeln und Automatisieren. Das Programmsystem läuft auf PCs und unterstützt zahlreiche Hardware-Zusätze, die die Anbindung des Programms an einen laufenden Prozess ermöglichen.

Ein **virtuelles Instrument** (**VI**) ist ein Anwenderprogramm. Es hat ein **Frontpanel** (Bedienoberfläche mit Schaltern, Lampen, Zahlen, Graphiken) und ein **Diagram** (Programm in Form eines Datenflussplanes unter Verwendung zahlreicher SubVIs). Ein SubVI ist einem

herkömmlichen Unterprogramm oder einer herkömmlichen Funktion adäquat, nur dass der Aufruf eine graphische Form erhalten hat.

1.1 Bestandteile eines LabViewprogramms (VI)

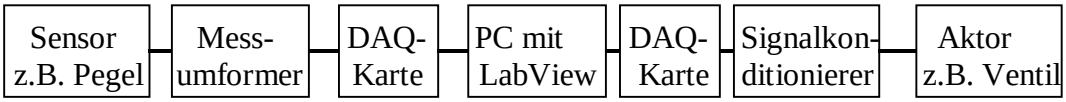
1. Das Frontpanel ist eine gefärbte Bildschirmfläche, auf der die Bilder von Schaltern, Drehknöpfen, Sollwertgebern, Schiebereglern, Zahleneingaben, LEDs, Balkenanzeigen, Zeigerinstrumenten, Zahlenausgabefenstern, Thermometern, Graphiken und weiteren Elementen der Bedienung eines Prozesses platziert werden. Alle Elemente sind interaktiv, d.h. sie reagieren in irgendeiner Weise auf die verschiedenen Maustasten.

2. Das Diagram ist ein Blockschaltbild oder Datenflussplan den Regeln und Elementen der graphischen Programmiersprache G folgend. Es gibt einfache Operationen, wie + – * / AND OR >0 = , aber auch Programmstrukturen wie Schleifen und Verzweigungen.

3. SubVIs sind Unterprogramme zum Zugriff auf Messfühler, Ventile, Dateien oder komplexe Funktionen, wie Kreiserkennung in der Bildverarbeitung. Der Nutzer kann auch eigene SubVIs programmieren.

1.2 DAQ (Data Aquisition, Datenerfassung)

Der Signalweg vom Messen bis zum Reagieren (Steuern oder Regeln) ist:



Sensor: Sensoren liefern Messsignale, z.B. PT100-Thermometer, Pegelmelder, Durchflussmesser, Leitfähigkeitsmesszelle, usw.
Messumformer: Wandeln und normieren Signale mittels Verstärkung, Nullpunkt- und Messbereicheinstellung, Linearisierung usw.
DAQ-Karte: Wandelt Spannungen bzw. Ströme entweder in logische 0-1-Größen (binäre oder digitale Eingänge) oder in Zahlen (analoge Eingänge). Bei der Ausgabe erfolgt die umgekehrte Wandlung von logischen Größen bzw. Zahlenwerten in Spannungen bzw. Ströme.
Signalkonditionierer: Bereiten Signale auf mittels Verstärkung, Halteglied, Erzeugung eines genormten Signals (4 bis 20 mA bzw. –10 bis +10 Volt bzw. 0 V / 24 V).
Aktor: Aktoren beeinflussen einen Prozess direkt. Aktor ist z.B. ein Ventil , eine Pumpe, eine Heizung usw. Jeder Aktor hat zusätzlich eine eigene Steuereinheit, die das Signal aus dem PC bzw. dem Signalkonditionierer verwendet, um den Aktor zu steuern (z.B. den gewünschten Ventilhub einzustellen oder eine gewünschte Pumpendrehzahl).

Andere Möglichkeiten der Messwertübertragung (Auswahl)

Buskarte, z.B. der GPIB (General Purpose Interface Bus) hat 24 parallele Leitungen (8 Daten-, 5 Steuer-, 3 Handshake- und 8 Masseleitungen). Er überträgt 8 Bits=1 Byte parallel. 30 Geräte mit den Adressen 1...30 sind anschließbar.

RS232: Serielle Schnittstelle (Modem). Datenbits werden nacheinander übertragen. Es ist nur ein Partner möglich. Die Übertragung ist zumeist langsam (im Schnitt etwa 50 kBit/s).

USB: ist ein bitserieller Bus, die einzelnen Bits des Datenpaketes werden also nacheinander übertragen. Etwa 300 MBit/s werden übertragen. Ein Strom von etwa 500 mA steht dem angeschlossenen Gerät zur Verfügung.

1.3 Ein einfaches LabView-Programm

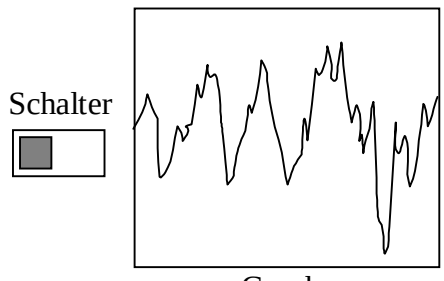
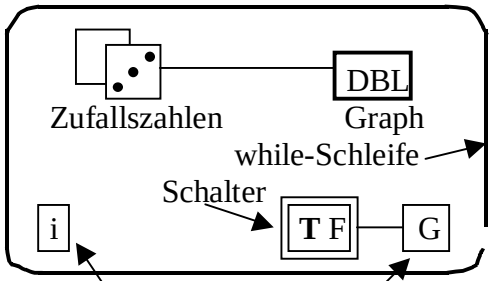
Toolspalette: Fenster mit verschiedenen Kursorsymbolen, von denen 4 für uns von Interesse sind. Ist das Fenster nicht sichtbar, klickt man "Windows" an, dann "Toolspalette".

1. Der Händchen-Kursor dient zum Steuern, Schalten und Regeln auf der Bedienoberfläche
2. Der Pfeilkursor dient zum Editieren des Frontpanels und zum Programmieren im Diagram
3. Der A-Kursor dient zum Beschriften oder Ändern von Texten in Textboxen
4. Der Drahtröllchen-Kursor dient im Diagram zum Verbinden der Elemente

Elementpalette: Sie wird beim Aufbau des Frontpanels benötigt. Man klickt mit der rechten Maustaste auf eine freie Stelle des Panels, dann erscheint die Palette. Über diese Palette wählt man Schalter, Drehknöpfe, LEDs, Zeigerinstrumente, Graphiken usw. aus und zieht diese Elemente an den Ort auf dem Frontpanel, wo man sie haben möchte.

Functionpalette: Sie wird beim Programmieren des Diagrams benötigt. Man klickt mit der rechten Maustaste auf eine freie Stelle des Diagrams, dann erscheint die Palette. Die Palette liefert Konstanten (Zahlen, Textboxen, TRUE/FALSE), Operationsboxen (+ – * / AND OR usw.), Programmstrukturen (Sequence aus Frames, Schleifen, Verzweigung), DAQ-Funktionen und mathematische Funktionen usw. Man zieht die Elemente in das Diagram und verbindet sie dann mit der Drahtrolle.

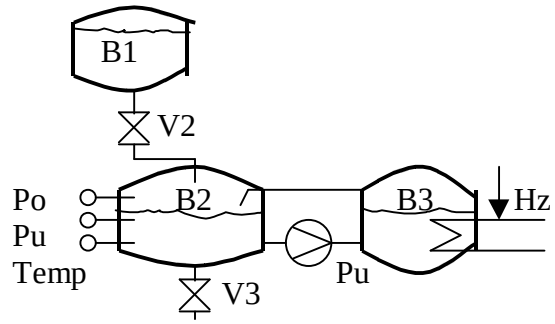
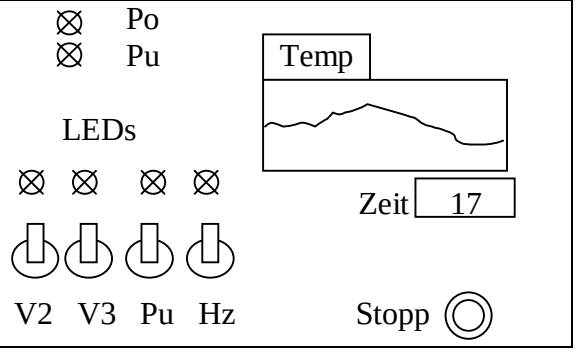
Programmbeispiel Zufallskurve (Rauschen) auf einem Graph.

Ansicht des Frontpanels (Bedienoberfläche)	Diagram (Graphische Programmierung)
	

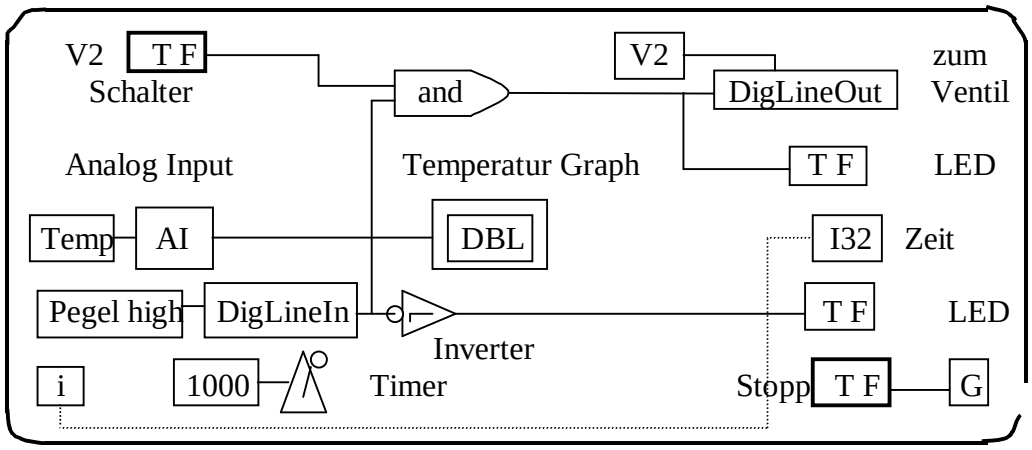
Wir starten das Programm mit RUN (⇒ , ein weißer Pfeil nach rechts in der Menüleiste oben über dem Frontpanel). Es gibt für spezielle Anwendungen auch ein wiederholtes Starten (Run continuously, ein Symbol aus zwei Pfeilen). Wir stoppen das Programm mit dem roten Stoppknopf, ebenfalls in der Menüleiste oben.

1.4 Beispiel Temperatursteuerung

Im AT-Labor von UV/BT steht eine kleine Versuchsanlage. Sie besteht aus einem Rechner mit DAQ-Board, einer Kabelverbindung zu den Signalkonditionierern (Black-Box) und einem Versuchsaufbau.

Versuchsaufbau mit 3 Behältern B1, B2, B3, 2 Ventilen V2 und V3, einer Kreislaufpumpe Pu, einer Heizung Hz, 2 Pegelmeldern Pu und Po und einem Thermometer Temp	Frontpanel auf dem PC mit 6 LEDs für die Pegelstände und Schalterstellungen, 4 Schaltern, einem Temperaturgraphen, Zeitangabe und Stopptaste
	

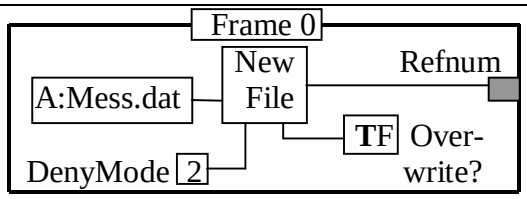
Die Pegelmelder in diesem Aufbau arbeiten invertiert, d.h. Sensor vom Wasser berührt liefert eine 0, Sensor frei liefert eine 1. Alle Signale kommen und gehen über DAQ-SubVIs, z.B. *DigLineIn* liefert über den Anschluss *line state* den Status des angeschlossenen Signals (0 oder 1). *DigLineOut* gibt z.B. über den Anschluss *line state* eine 0 oder eine 1 als Spannung an ein Ventil weiter. Das Diagram ist in der folgenden Graphik zu sehen:



Wir sehen eine große while-Schleife, die alle 1000 ms einmal durchlaufen wird. Mit einer Stopptaste kann sie abgebrochen (terminiert) werden. Eine UND-Logik verhindert, dass trotz Schalterstellung 1 des V2-Schalters links oben kein Signal an die Funktion *DigLineOut* gegeben wird, wenn der *Pegel high* eine 0 meldet, d.h. der obere Pegel berührt wird. Vom Kanal *Temp* wird das Temperatursignal mittels AnalogInput (*AI*) eingelesen und direkt an die Graphik weitergereicht. Das Pegel-high-Signal wird invertiert an die LED "Po" weitergegeben (leuchtet also bei Berührung). Die Zeitangabe kommt aus dem Schleifenzähler i (I32 ist der Zahltyp Integer 32 Bit des verlangten Wertes).

1.5 Messwertausgaben, z.B. Temperaturwerte auf eine Datei schreiben

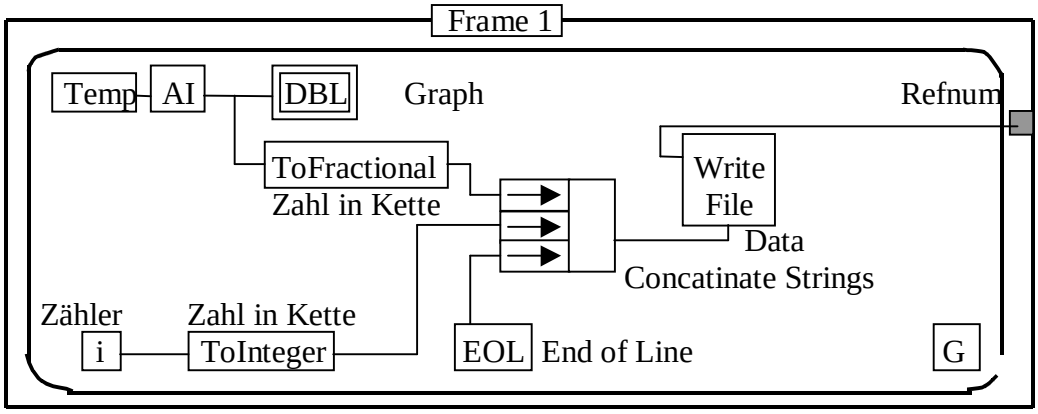
Wir benötigen einen Programmaufbau aus 3 Teilprogrammen, die nacheinander ablaufen müssen (eine Sequence mit 3 Frames). Teil 1 (Frame 0) öffnet die Messdatendatei:



New File hat hier 3 Argumente:
Pfadname der Datei ist A:Mess.dat
DenyMode 2 erlaubt Lesen und Schreiben
Overwrite TRUE erlaubt das Überschreiben
Refnum leitet das Dateihandle weiter

Die Weitergabe des Dateihandles (Refnum) erfolgt über einen Datentunnel (Sequence local) in all nachfolgenden Frames.

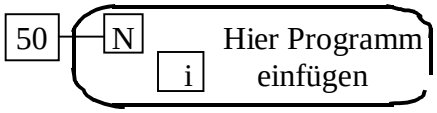
Im Frame 1 (unser eigentliches Regel- und Steuerprogramm) müssen die Binärzahlen in Zeichenketten umgewandelt werden, zu einer Zeile mit EOL-Endekennung zusammengesetzt und dann ausgegeben werden. Die Graphik zeigt nur die Elemente aus Frame 1 und der while-Schleife, die unmittelbar der Datenausgabe dienen:



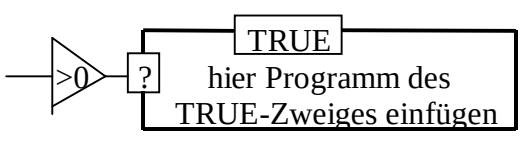
Die Temperaturdaten werden angezapft, und mit der Funktion *ToFractional* in eine lesbare Zahl, d.h. in eine Zeichenkette, verwandelt. Der Integer-Schleifenzähler i wird mit der Funktion *ToInteger* ebenfalls in eine Zeichenkette umgewandelt. Beide Zeichenketten werden mit der Funktion *Concatinate Strings* zusammengefasst und durch das Zeichen EOL abgeschlossen. Die so aufbereitete Datenzeile wird von der Funktion *Write File* auf die Datei mit dem Dateihandle *Refnum* ausgegeben. Refnum stammt aus Frame 0.

Im letzten Frame, Frame 2, ist lediglich noch die Funktion *File Close* an das Dateihandle Refnum anzuschließen. Die Messdatendatei wird geschlossen.

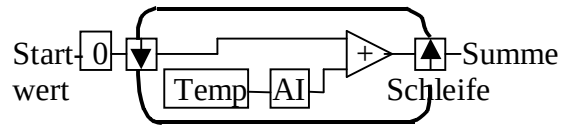
1.6 Weitere Programmstrukturen außer while-Schleife und Sequence



Das Programm im Inneren der **for-Schleife** wird hier 50 mal durchlaufen. Die Werte des abgreifbaren Schleifenzählers i sind nacheinander i = 0, 1, 2, 3, ..., 49

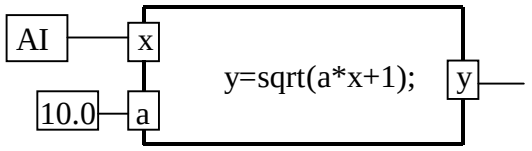


case (oder Verzweigung oder if-Anweisung) hat einen TRUE- und einen FALSE-Zweig, von denen einer leer sein darf. Welcher Zweig durchlaufen wird, bestimmt die Logik am "?"



Schieberegister entsprechen einfachen Variablen in C. Ein in der Schleife berechneter Wert wird rechts ausgeschleust, aber gleichzeitig auch links wieder eingeschleust.

Zum Startwert, der vor dem ersten Schleifendurchlauf 0 gesetzt wurde, wird bei jedem Schleifendurchlauf ein Temperaturwert der Analogeingabe AI hinzuaddiert. Rechts kann man den Augenblickswert der Summe abgreifen. Nach dem letzten Schleifendurchlauf ist es der Endwert der Temperatursumme.



Formelknoten gestatten auch kompliziertere Berechnungen. Operanden sind die aus C, z.B. || && != == + – * / ++ , aber auch mathematische Funktionen, wie sin, cos, sqrt, log, exp usw.

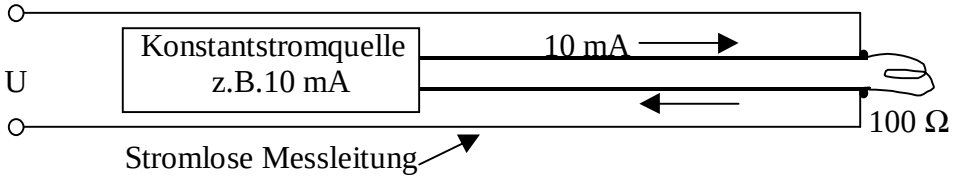
Links gehen in den Beispiel-Formelknoten die Werte x and a ein. Wert x ist eine Analogeingabe, z.B. eine Temperatur, a ist hier eine Konstante (10,0). Im Formelknoten wird hier nur ein y berechnet, es wären aber mehrere Ausgabegrößen möglich. Das berechnete y wird weitergegeben.

Arrays sind Vektoren und Matrizen aus Elementen gleichen Typs (gleichen Speicherbedarfs). **Cluster** entsprechen den *Strukturen* in C, und sind Datenansammlungen unterschiedlichen Typs.

Zeichenkettenfunktionen dienen der Textverarbeitung, um z.B. Ausgabezeilen von Dateien oder Fehlermeldungen aufzubereiten. Wichtige Funktionen sind z.B. Stringlänge bestimmen, Zeichenketten verknüpfen (Concatinate), Zahlen in Zeichenketten umwandeln unter Verwendung von Format-Strings, wie "%5i" oder "%10.3f". Auch Datum-Uhrzeit-Strings können in unterschiedlicher Form für den Einbau in Ausgabetexte angefordert werden.

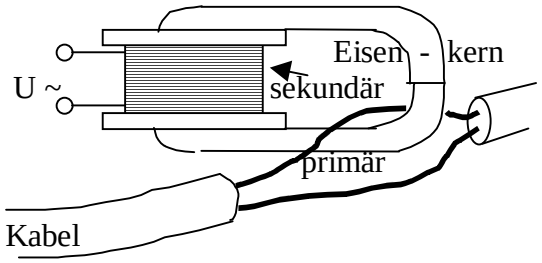
Auswahl einiger Sensoren im AT-Praktikum von UV/BT

Ein **PT100**-Platindraht-Widerstandsthermometer besteht aus einer gekapselten Platindrahtwicklung mit 100 Ω Widerstand bei 0 °C. Eine mögliche Messanordnung benutzt z.B. eine Konstantstromquelle, die mit 12 V gespeist wird. Diese passt die Spannung U an den Widerstand an, damit sich die Stromstärke sich nicht ändert. Diese sollte klein genug sein, damit der Platinwendel nicht zum Heizwendel wird.

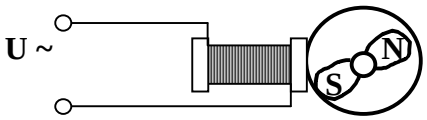


$U = R \cdot I$ $U \approx 100\ [\Omega] \cdot 0{,}01\ [A] = 1{,}0\ [V]$ $L = U \cdot I = 1{,}0\ [V] \cdot 0{,}01\ [A] = 0{,}01\ [W]$

Die **Wechselstromzange** misst die Stärke eines durch ein Kabel fließenden Wechselstroms nach dem Tranformatorprinzip: Eine halbe Primärwindung, 1000 oder mehr Sekundärwindungen. Die Stromstärke erscheint als Wechselspannung U am Ausgang, z.B. 1 Volt entspricht 10 Ampere Stromstärke im Kabel. Dieses muss jedoch geteilt werden.



Das **Windrad (Anemometer)** misst Windgeschwindigkeiten bis v=20 m/s. Die magnetisierten Flügel erzeugen in einer Spule eine kleine Wechselspannung als U als Signal.



Der **Prema-Messcomputer** vereint Multimeter und PC und kann elektrische Größen messen.

Gleichspannung	±100 mV	bis	±1000 V	mit R>10 MΩ, Fehler<10µV
Widerstand	100 Ω	bis	10 MΩ	mit I = 1 mA
Wechselspannung	100 mV	bis	700 V	
Stromstärke		bis	2 A	

Temperatur	mit	PT25, PT100, ..., PT100	
	mit Thermoelement	Typ J (Fe - CuNi)	-210 bis 1200 °C
	mit Thermoelement	Typ B (Pt30Rh - Pt16Rh)	42 bis 1820 °C

Messstellenumschalter 4-polig immer 1 aus 20 möglichen Kanälen
AD-Wandlung integrierend (Kondensator wird aufgeladen)
Messzeiten sind immer ganze Vielfache von 20 ms (Netzperiode)

Menü	Funktion	Gleichspannung, Wechselspannung, Widerstand, Temperatur, ...			
	Bereich	100 mV	1 V	10 V	.....
	Messzeit	20 ms	40 ms	...	100 s
	Sonstiges	Wahl zwischen Zahlenanzeige oder Diagramm			
		Messstellenumschalter programmieren			
		Messwertspeicher (Kopieren, begrenzen, ...)			
		.....			

Unsere **Beispielmessung** mit dem Prema-Messcomputer im Labor: 3 PT100 messen alle 7 Sekunden die Temperatur in einem Glaskolben. 200 Messwerte je Thermometer werden gespeichert. Es entstehen 4 Dateien: 3 Zahlenkolonnen mit den Temperaturwerten der 3 Thermometer und eine Headerdatei mit allgemeinen Angaben, z.B. die Intervallzeit der Messung. Programm DASY macht uns eine ASCII-Datei daraus mit der Form

Zeit	AA	BB	CC
7	24.3	24.3	24.4
14	26.5	26.5	26.5
..	.....	.....	.....

Der **Micromec Datenlogger** (Fa. Micromec aus Freiburg) erlaubt Akkugestütztes Messen und Speichern der Daten vor Ort, z.B. im Kühlhaus oder in einer Lagerhalle. Zum Datenlogger gibt es zahlreiche unterstützte Sensoren, z.B.

Druck	350 mbar	bis	10 bar
Temperatur	PT100,	Thermoelemente, z.B. NiCr - Ni	
Feuchte			
Windgeschwindigkeit			
pH			
Strom, Spannung			

Ein Messprogramm beim Micromec-Datenlogger (Taste B) umfasst:
Kontrolle der angeschlossenen Sensoren
Datum und Uhrzeit des Messbeginns
Messdauer und Intervallzeit
mit oder ohne Glättung (Mittelwertbildung)

Unsere **Beispielmessung** im Labor mit dem Micromec-Datenlogger: Ein Haarfön bläst ein Windrad und ein PT100 an. Stromstärke am Fön, Windgeschwindigkeit und Temperatur werden alle Sekunde gemessen.

2. Relationale Datenbanken und SQL

Literatur: Matthiessen & Unterstein, Addison-Wesley 1997

Vorteile von Datenbanken:
- Es gibt keine doppelt gespeicherten Information, so dass keine Abweichungen auftreten können, z.B. unterschiedliche Adressen eines Kunden
- Man kann wichtige Daten gegen versehentliches Löschen sperren
- Man hat zahlreiche Auswertemöglichkeiten (Kundenliste, Artikelliste, ...)

SQL (Structured Query Language) gestattet die Abfrage der Datenbank vollautomatisch aus anderen Programmen heraus, z.B. Kundenadressen für Serienbriefe in MS-Word.

2.1 Grundlagen einer relationalen Datenbank

Domänen sind den Typen in C vergleichbar:

CARDINAL	0, 1, 2, 3, ...
NUMERIC	-3.14, 2.7119
STRING	Senfgurke
Datum	24.06.07
.....	.....

Operationen auf Domänen:

Add, Mul, ...	auf	CARDINAL
Sin, Cos, ...	auf	NUMERIC usw.

Definitionen sind den Zuweisungen vergleichbar:
(werden auch *Tupel* genannt)

Name	=	"Stein"
Größe	=	184 cm

(*Bezeichner* , *Attribut*)

Ein **Relationen-Schema** ist die Eingabe-Maske oder der Tabellenaufbau aus Spaltenbezeichnern und Datentyp. Die Tabelle selbst wird **Relation** genannt.

Ein **Datenbank-Schema** besteht dann aus:
- einer Menge von Bezeichnern für Relationen (Tabellenliste)
- zu jeder Relation (Tabelle) das Relationen-Schema (der Tabellenaufbau)
- zu jeder Relation (Tabelle) Bedingungen (Schlüsselnummern)

Die Komponenten eines **Datenbanksystems** (DBS) sind:
- ein System zur Veränderung des Datenbank-Schemas selbst, falls das gewünscht wird
- Möglichkeiten der Dateimanipulation (Hinzufügen von Daten, Korrigieren, Löschen, ...)
- ein Reportgenerator für Listen, Tabelle, Auszüge, Statistiken, ...
- eine Zugriffssprache für andere Anwendungen (SQL)

Weitere Begriffe:

Entität: Ein eindeutiges Objekt, z.B. eine Person oder ein KFZ oder ein Bestellschreiben
Abhängige Entität: Eindeutiges Teil einer Entität, z.B. eine Position einer Bestellung
Schlüssel: Eindeutige Nummerierung innerhalb einer Relation (in einer Tabelle), z.B. die Bestellnummer oder das Kfz-Kennzeichen.
Normalisierung: Daten treten nur einmal auf, z.B. Kundendaten (Name, Telefon, Adresse, ...) nur in der Kundenliste (Kundentabelle)

Relationenalgebra: Dient der Kombination und Auslese von Zeilen aus Tabellen

$\times$	Kreuzprodukt	$(1, 3, 7) \times (A, y) = (1A, 1y, 3A, 3y, 7A, 7y)$
$\cup$	Vereinigung	$(1, 3, 7) \cup (2, 3) = (1, 2, 3, 7)$
$\cap$	Durchschnitt	$(1, 3, 7) \cap (2, 3) = (3)$
\	Differenz	$(1, 3, 7) \setminus (2, 3) = (1, 7)$

2.2 Die Datenbank ACCESS ®

Literatur: Albrecht & Nicol, Microsoft Access, 2001

Access arbeitet mit Tabellen aus Zeilen und Spalten. Eine Zeile ist z.B. ein Artikel im Lager, Spalten sind z.B. Artikelnummer, Regal, Preis, Anzahl, ...

Beispiel Datenbank **GRUEN** mit den Tabellen und ihren Tabellenspalten

Gattung:	GNr,	GatBez,	BotanischerName		
Ernte:	ENr,	PflNr,	EDatum,	EOrt,	GNr
Pfluecker:	PflNr,	Name,	Vorname,	Semester	
Blattdaten:	BlNr,	ENr,	LaengeMM,	BreiteMM,	Stachelzahl, Gruenton

2.2.1 Wir wollen eine neue Datenbank mit Assess erstellen:

-> Start -> alle Programme -> Microsoft Office -> Access -> Neue leere Datenbank -> Namen ändern in ***Gruen***.accdb
-> Erstellen (in der Menüleiste ganz oben) -> Tabellenvorlagen -> Wir nehmen immer Vorlage ***Kontakte*** : Durch rechten Mausklick auf eine Tabellenspalte können wir die Spalte umbenennen.

ID	umbenennen in	GNr	
Firma	umbenennen in	GatBez	usw.

Die restlichen Spalten löschen wir (Ausnahme die letzte Spalte *Neues Feld einfügen*).
Nach dem Umbenennen und Spaltenlöschen geben Sie gleich ein paar Daten ein. In der ersten Spalte nicht. Die Schlüsselnummern dort werden automatisch vergeben. Nach der Eingabe haben Sie etwa die folgende Tabelle:

GNr	GatBez	BotanischerName
1	Liguster	Ligustrum ovalifolium
2	Buche	Fagus Sylvatica

Wir beenden die Eingabe mit einem Klick auf das × der Tabelle , antworten mit *Ja* auf die Frage „ *Speichern?*“ und benennen die Tabelle anschließend ***Gattung***.

2.2.2 Nach dem gleichen Schema legen wir die Tabellen *Pfluecker* an. Wir nehmen wieder ***Kontakte*** als Vorlagetabelle: Wir löschen die Spalte *Firma* und benennen dann um. Die restlichen Spalten löschen wir bis auf die letzte Spalte *(Neues Feld einfügen).*

ID ⇓	Nachname ⇓	Vorname ⇓	E-Mail-Adresse ⇓
PflNr	Nachname	Vorname	Semester
1	Fleischer	Fips	A1
2	Last	James	B5

Wir beenden die Eingabe mit einem Klick auf das × der Tabelle, antworten mit *Ja* auf die Frage „ *Speichern?*“ und benennen die Tabelle anschließend ***Pfluecker*** .

2.2.3 Nach dem gleichen Schema legen wir die Tabelle *Blattdaten* an.

ID ⇓	Firma ⇓	Nachname ⇓	Vorname ⇓	E-Mail-Adresse ⇓
BlNr	ENr	LaengeMM	BreiteMM	Stachelzahl
1	1	47	21	0
2	1	52	28	0

Wir beenden die Eingabe mit einem Klick auf das × der Tabelle, antworten mit *Ja* auf die Frage „ *Speichern?*“ und benennen die Tabelle anschließend ***Blattdaten*** . Wir wollen jetzt zusätzlich eine Spalte ***Gruenton*** mit fest vorgegebenen Eintragungen einfügen: -> Öffnen Blattdaten -> rechter Mausklick auf die letzte Spalte mit der Bezeichnung "*Neues Feld einfügen*" -> *Nachschlagespalte* anklicken -> "*selbst Werte eingeben*" anklicken -> weiter -> neben dem * beginnend untereinander die drei Grüntöne *hell, mittel, dunkel* eintippen -> weiter -> umbenennen der Spalte in ***Gruenton*** -> fertigstellen -> Klick auf das × der Tabelle. Wir öffnen die Tabelle ***Blattdaten*** erneut und legen in jeder Zeile einen Grünton fest, indem wir den PopUp-Pfeil der Zelle drücken und eine Auswahl treffen, z.B. *hell*.

2.2.4 Nach dem gleichen Schema legen wir die Tabellen *Ernte* an. Wir nehmen wieder ***Kontakte*** als Vorlagetabelle:

ID ⇓	Firma ⇓	Nachname ⇓	Vorname ⇓	E-Mail-Adresse ⇓
ENr	PflNr	EDatum	EOrt	GNr
1	1	24.11.09	Volksbank	1
2	1	12.12.09	Stadtpark	1

Wir beenden die Eingabe mit einem Klick auf das × der Tabelle , speichern und nennen unsere Tabelle ***Ernte*** .

2.2.5 Beziehungen zwischen den Tabellen festlegen:
Beziehungen legen fest, wie die einzelnen Tabellen miteinander verknüpft sind. Fast immer läuft die Verknüpfung über Schlüsselnummern. In unserer kleinen Datenbank haben wir die Schlüsselnummern ENr, GNr, PflNr, die für eine Verknüpfung der vier Tabellen sorgen.
-> Datenbanktools -> Beziehungen -> Taste Shift und Pfeiltaste unten, bis alle 4 Tabellen im PopUp-Fenster markiert sind -> Hinzufügen -> Schließen -> ENr in Tabelle *Blattdaten* klicken und nach ENr von Tabelle *Ernte* ziehen. Es erscheint das PopUp-Fenster „Beziehung bearbeiten“. Wir klicken auf den Button „*erstellen*“. Auf die gleiche Weise GNr mit GNr und PflNr mit PflNr verbinden. Jetzt sind die drei Tabellen *Blattdaten, Pfluecker* und *Gattung* jeweils mit der Tabelle *Ernte* verknüpft. Wir beenden die Eingabe mit einem Klick auf das × , speichern mit *Ja*.

2.2.6 Autobericht der Blattdaten: Tabelle *Blattdaten* anklicken -> Erstellen -> Bericht: Es erscheint der Bericht in Form einer Tabelle -> externe Daten -> Excel -> Pfadnamen auf X-Platte legen und Ordner IV -> OK.

Wir schauen uns den Bericht mit Hilfe von Excel an. Wir löschen die erste Zeile mit dem Datum, schieben die Spalten eng zusammen, markieren den Tabellenbereich, kopieren und fügen die Kopie in die Hausarbeit ein als Beweis, dass eine Datenbank ***Gruen*** angelegt wurde.

3. Einführung in Excel ® (Tabellenkalkulation)

Literatur: Said Baloui, Excel, Markt&Technik

Excel kann rechnen, Graphiken erzeugen, Datenbank sein.

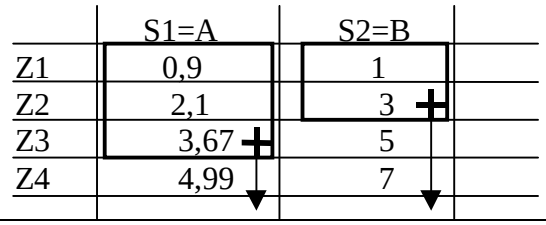
3.1 Einfache Beispiele zu Excel

Eine Tabelle besteht aus **Feldern**, die in **Zeilen** und **Spalten** angeordnet sind. **Selektieren** heißt ein Feld anzuklicken bzw. einen **Bereich** (dieser wird schwarz, aber das obere linke Feld weiß und dick umrandet). Die Entertaste wird hier im Skript mit **E** abgekürzt. Die Shift-Taste ist die Großschreibtaste direkt über Strg.

	S1=A	S2=B	S3=C	.....
Z1	***3,14***			
Z2			**-0,5997**	
Z3		***-9,3***		

Wir selektieren ***A1*** und tippen ***3,14*** ein.
Wir selektieren ***B3*** und tippen ***-9,3*** ein.
Wir selektieren ***C2*** und tippen und selektieren
sel. sel.
=cos(***A1*** * ***B3***) **E**
Auf C2 erscheint das Ergebnis. Ändern wir die Zahl in A1, ändert sich auch das Ergebnis.

- Man kann selektierte Zeilen/Spalten/Bereiche **löschen, kopieren**.
- Man kann vor einer selektierten Zeile/Spalte leere Zeilen/Spalten **einfügen**
- Man kann sel. Zeilen/Spalten/Bereichen **andere Formate** verpassen, z.B. "Datum"
- Man kann eine selektierte Zahlenreihe oder Datumsreihe durch ***Ziehen*** verlängern.



	S1=A	S2=B
Z1	0,9	1
Z2	2,1	3
Z3	3,67	5
Z4	4,99	7

Beispiel Spalte S1=A: Es erscheinen die Werte der Ausgleichsgeraden, die durch die eingerahmten y-Werte definiert ist. Die 3 x-Werte sind 1, 2, 3, dann fortgesetzt 4, 5, 6, ... **Beispiel Spalte S2=B:** Es erscheinen die Werte der Zahlenfolge 1, 3, 5, 7, 9, 11, ...

Die **Rechen- und Vergleichsoperatoren** nach fallender Priorität geordnet sind:

^ (hoch) * / + – < <= = >= > <>(ungleich)

Funktionen finden wir, wenn wir das Ikon f_x anklicken. Eine Beschreibung der Funktion finden wir, wenn wir im Fenster der Funktion das Fragezeichen ***?*** anklicken.

3.2 Excel als Datenbank

Für einfachere Aufgaben kann Excel gut als Datenbank verwendet werden, z.B. für Adressdateien (Serienbriefe, Rechnungslegung) oder für Versuchsdaten, die direkt aus der Messapparatur übertragen werden.

Beispiel Datenbank Grün

Arbeitet man mit Schlüsselnummern, die später eventuell das Filtern oder den Export der Daten in andere Auswerteprogramme erleichtern, dann legt man irgendwo, z.B. im oberen Teil der Mappe, kleine Tabellen an, in denen die Schlüsselnummern erklärt werden. Beispiele sind Gattung, Pflücker, Ernte, Grünton. Die eigentlichen Daten bilden dann eine einzige große, zusammenhängende Tabelle, hier die Blattdaten auf den Zellen A15:F29.

Viele Anwender verzichten jedoch auf eine Verschlüsselung, d.h. bei Pflücker würde immer der Name des Pflückers stehen, z.B. *Adler*, und nicht seine Pflückernummer, die *1*. Das sieht jedoch anders aus, wenn kategoriale Daten anliegen, z.B., wenn die Menge des im Blatt gespeicherten Chlorophylls mit dem Grünton wächst. In diesem Falle könnte man die Spalte Grünton direkt in einer Regressionsanalyse als Einflussvariable verwenden, was bei nominalen Daten (z.B. hell, mittel, dunkel) nicht geht. Hier sind die Nummern demnach hilfreich. Man muss sie jedoch im Format *Zahl* anlegen.

	A	B	C	D	E	F
1	Gattung	GNr	Bot. Name			Grünton
2	Hainbuche	1	Carpinus betulus			1=hell
3	Kastanie	2	Castanea			2=mittel
4	Birke	3	Betula			3=dunkel
5						
6	Pflücker	Vorname	Semester	PflNr		
7	Adler	Adelheid	S2	1		

8	Biber	Bernhard	S4	2		
9						
10	Erntedatum	Ort	ErnteNr	Pflücker	Gattung	
11	22.8.2009	Volksbank	1	2	1	
12	27.9.2009	Mautepark	2	1	1	
13	1.10.2009	Parkplatz	3	3	3	
14						
15	BlattNr	Länge mm	Breite mm	Grünton	Gattung	Ernte
16	1	67	34	2	1	1
17	2	69	39	2	1	1
18	3	58	31	2	1	1
19	4	63	30	2	1	1
20	5	67	35	2	1	1
21	6	72	37	2	1	1
22	7	44	27	1	3	3
23	8	47	25	1	3	3
24	9	39	26	1	3	3
25	10	52	23	1	3	3
26	11	48	29	1	3	3
27	12	46	31	1	3	3
28	13	50	28	1	3	3
29	14	37	25	1	3	3

Wir wollen mittellange Blätter herausfiltern. Dazu selektieren wir alle Blattdaten A15:F29, dann klicken wir auf Daten --> Filter --> Autofilter --> Popup-Pfeil bei der Datenreihe *Länge* --> benutzerdefiniert. Es erscheint eine 2x2-Tabelle, der wir durch simples Anklicken links bzw. Eintippen rechts folgendes Aussehen geben:

Ist größer oder gleich	50
Ist kleiner oder gleich	60

Wir geben OK. Das Ergebnis ist die folgende Auswahl, die aber auf den Zeilen 15-18 steht, d.h.. auf dem Platz unserer Originaldaten. Dort kann die Auswahltabelle nicht bleiben.

15	BlattNr	Länge mm	Breite mm	Grünton	Gattung	Ernte
16	3	58	31	2	1	1
17	10	52	23	1	3	3
18	13	50	28	1	3	3

Wir selektieren die Auswahltabelle A15:F18, klicken auf Kopieren. Dann selektieren wir ein Feld exakt derselben Größe, z.B. die Zellen A41:F44, und fügen die Kopie dort ein (--> Bearbeiten --> Einfügen bzw. das Symbol mit dem Koffer). Dann drücken wir die Taste Esc, damit der flimmernde Rahmen der Auswahl verschwindet. Wir haben die Auswahl jetzt als separate Tabelle.

41	BlattNr	Länge mm	Breite mm	Grünton	Gattung	Ernte
42	3	58	31	2	1	1
43	10	52	23	1	3	3
44	13	50	28	1	3	3

Die alte, vollständige Tabelle stellen wir wieder her, indem wir unsere Originalauswahl mit den Popup-Markierungen auf den Zellen A15:F18 nochmals selektieren, dann --> Daten --> Filter --> alle anzeigen. Die Popup-Pfeile verschwinden durch --> Daten --> Filter --> Autofilter (das Häkchen sozusagen wieder wegklicken). Jetzt steht einer Weiterverarbeitung der ausgewählten Daten nichts mehr im Wege.

3.3 Bedingte Berechnungen und Fehlwerte in Excel

Ähnlich den if-Anweisungen möchte man bestimmte Alternativen bei der Berechnung von Werten haben. Excel bietet das an.

	.....	C	D	E
Z1		1	4	0
Z2		3	2	0
Z3		9	1	1
Z4		4	5	0
Z5		2	6	0

Beispiel bedingte Berechnungen
Selektiere E1 und tippe ein:
=wenn(und(*sel.C1***>5);***sel.D1***<4); 1; 0) Enter**
Auf E1 erscheint das Ergebnis 0
Durch Ziehen erscheinen weitere Ergebnisse auf den Feldern E2, E3 usw.

Wenn C1>5 ist **und** D1<4 ist, dann nimm die **1** (den ersten Ausgabewert), sonst die **0** (zweiten Ausgabewert). Statt 1 und 0 können auch Felder selektiert werden mit Werten, die nach zwei verschiedenen Formeln zuvor berechnet wurden.
Die Funktion **WENN** hat den formalen Aufbau WENN (logische Bedingung ; Ja-Ausdruck ; Nein-Ausdruck)
Die log. Funktion **UND** hat den Aufbau UND(log.Wert 1; log.Wert 2; ...; log.Wert n) und liefert nur den Wert *wahr*, wenn alle logischen Werte *wahr* sind.
Die log. Funktion **ODER** hat den Aufbau ODER(log.Wert 1; log.Wert 2; ...; log.Wert n) und liefert nur den Wert *falsch*, wenn alle logischen Werte *falsch* sind.

In manchen Berechnungen berücksichtigt Excel **fehlende Werte**, in vielen dagegen nicht. Fehlende Werte sind nicht eingetippte oder nicht berechnete Werte, d.h. leere Felder. Beispiel Summe:

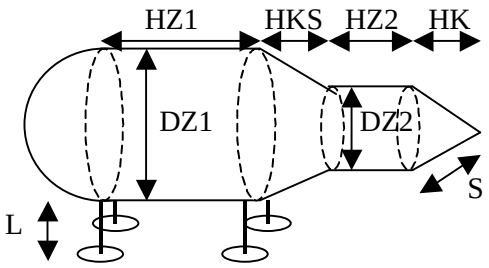
	A	B	C	D	E	F	G		I
Z1	2	7		4					13

Selektiere *I1* und tippe ein
=summe(*sel.A1:G1***) Enter**

Trotz 4 fehlender Werte wird von Excel die Summe auf I1 richtig berechnet

3.4 Berechnung eines Behälters

Von einem Behälter (Silo, Reaktor, Apparat, usw.) sollen Oberfläche, Volumen, Leermasse, Füllmasse und Gesamtmasse berechnet werden.

Reaktorgefäß für die Hydrierung von Alkenen Hz1 = Höhe Zylinder 1 HKS = Höhe Kegelstumpf HZ2 = Höhe Zylinder 2 HK = Höhe Kegel DZ1 = Durchmesser Zylinder 1 DZ2 = Durchmesser Zylinder 2 L = Stützenlänge S = Länge der Kegelmantellinie	

Einige Berechnungsformeln (Radius R = D/2, H=Höhe, Ru = unterer Radius, Ro = oberer Radius, Hks = Höhe Kegelstumpf, Hsp = Höhe der Spitze, Ssp = Mantellinie der Spitze)

Körper bzw. Fläche	Volumen	Fläche
Kreisscheibe		$A=\pi R^2$
Quader mit L, B, H	V = L B H	A = 2 (L B + L H + B H)
Kugel	$V=\frac{4\pi}{3}R^3$	$A=4\pi R^2$
Zylinder	$V=\pi R^2 H$	$A=2\pi R H$ (nur Mantel)
Kegel	$V=\frac{\pi}{3}R^2 H$	$A=\pi R S \quad mit \quad S=\sqrt{R^2+H^2}$ (nur Mantelfläche)
Kegelstumpf Gegeben R_U, R_O, H_{KS} Hilfsgröße H_{SP} ist die Höhe der Spitze H=Gesamthöhe	$H_{SP}=\frac{R_O H_{KS}}{(R_U-R_O)}$ $H=H_{KS}+H_{SP}$ $V=\frac{\pi}{3}\left(R_U^2 H-R_O^2 H_{SP}\right)$	$S=\sqrt{R_U^2+H^2}$ $S_{SP}=\sqrt{R_O^2+H_{SP}^2}$ $A=\pi\left(R_U S-R_O S_{SP}\right)$ (nur Mantelfläche)

Es folgt die Exceltabelle. Einzige Berechnungsspalte ist hier Spalte B. Alles andere ist Kommentar.

	A	B	C	D	E	
1						
2						
3	HZ1	5	Höhe des	Zyl 1		Eingabedaten
4	HKS	2	Höhe des	Kegelst.		
5	HZ2	3	Höhe des	Zyl 2		
6	HK	1,5	Höhe des	Kegels		
7	D1	3,5	Durchmes	Zyl 1		
8	D2	2	Durchmes	Zyl 2		
9	L	1,5	Länge	einer	Stütze	
10						
11	Rho_St	7840	Dichte	Stahl	Kg/m^3	Eingabedaten
12	d	0,008	Blech-	dicke	m	

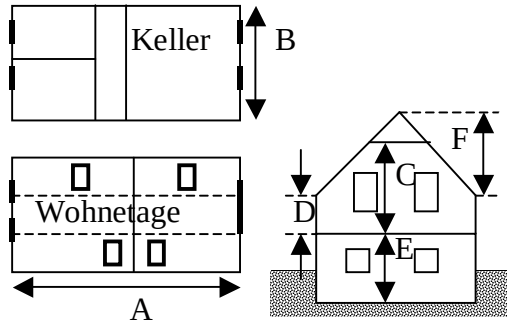
13	Stützgw	78	Gewicht	Stütze	Kg/m	
14	Tellergw	7,5	Gewicht	Teller	Kg	
15	Rho_Fü	1050	Dichte	Füllung	Kg/m^3	

Ab hier erfolgen die Berechnungen. Die von Excel berechneten Zahlenwerte sind in der B-Spalte zu sehen. Die Formeln, die erst beim Anklicken der Zelle sichtbar werden, sind hier im Skript ganz rechts zu sehen. (Abkürzungen: KS = Kegelstumpf, Sp = Spitze)

16						Hier in dieser Spalte stehen die Formeln aus der B-Spalte
17	R1	1,75	Radius	Zu D1		=B7/2
18	R2	1	Radius	Zu D2		=B8/2
19						
20	AHK	19,24	Fläche	Halb-	kugel	=2*PI()*B17^2
21	VHK	11,22	Volumen	Halb-	kugel	=2*PI()*B17^3/3
22	AZ1	54,98	Fläche	Zyl 1		=PI()*B7*B3
23	VZ1	48,11	Volumen	Zyl 1		=PI()*B17^2*B3
24	HSP	2,6667	Höhe	Spitze	KS	=B18*B4/(B17−□B18)
25	H	4,1667	Höhe KS	plus	Spitze	=B6+B24
26	S	4,5192	Mantel-	Linie	KS+Sp	=wurzel(B17^2+B25^2)
27	SSP	2,8480	Mantel-	Linie	Sp KS	=wurzel(B18^2+B24^2)
28	AKS	15,899	Fläche	Kegel-	stumpf	=PI()(B17*B26□−B18*B27)
29	VKS	10,570	Volumen	Kegel-	stumpf	=PI()*(B17^2*B25−□B18^2*B27)/3
30	AZ2	18,850	Fläche	Zyl 2		=PI()*B8*B5
31	VZ2	9,4248	Volumen	Zyl 2		=PI()*B18^2*B5
32	SK	1,8028	Mantel-	Linie	Kegel	=wurzel(B18^2+B6^2)
33	AK	5,6636	Fläche	Kegel		=PI()*B18*B32
34	VK	1,5708	Volumen	Kegel		=PI()*B18^2*B6/3
35						
36	Ages	114,63	Gesamt-	fläche		=B20+B22+B28+B30+B33
37	Vges	80,896	Gesamt-	volumen		=B21+B23+B29+B31+B34
38	MStütz	498	Masse	Stützen+	Teller	=4*(B9*B13+B14)
39	Mleer	7687,7	Leer-	masse		=B11*B36*B12+B38
40	Mfüll	84941	Masse	Füllung		=B37*B15
41	Mges	92629	Gesamt-	masse		=B39+B40

3.5 Beispiel Wärmeverbrauch

Der Wärmeverbrauch eines Hauses lässt sich grob berechnen, wenn man die Maße des Hauses hat, die Innen- und Außentemperatur sowie die Wärmedurchgangszahlen für das Mauerwerk, die Fenster und die Türen. Nicht berücksichtigt werden hier in dieser Berechnung der Luftaustausch und die Bereitstellung von Warmwasser.

Unterkellertes Siedlungshaus	
A=10 m Länge des Hauses B=6 m Breite des Hauses C=2,40 m Deckenhöhe Wohnetage D=0,90 m Kniestock E=2,10 m Deckenhöhe im Keller F=3,50 m Dachhöhe ab Kniestock Keller 4 Fenster a 0,8 m^2 Wohnetage 6 Fenster a 1,2 m^2, 1 Fenster 2 m^2 2 Außentüren a 2,1 m^2	

Wir benennen Sie die Maße der Geschosse mit Buchstaben, z.B. A, B, ...,E. Im Keller sind 14°C, in der Wohnetage 22°C. Das Spitzdach hat Außentemperatur. Diese sei im Jahresmittel 9°C.
Der Wärmedurchgangswert des Mauerwerks sei U = 0,23 W/(K· m^2), der U-Wert der Dachhaut oberhalb des Kniestocks und der Etagendecke ist U = 0,18 W/(K· m^2).

Wir berechnen in einer Exceltabelle die Wärmeabgabeflächen jedes Geschosses getrennt nach Mauerwerk und Fensterfläche. Wärmefluss in Watt ist $\dot{Q} = U \cdot A \cdot (\vartheta_i - \vartheta_a)$. Dabei ist A die Fläche mit Wärmedurchgangszahl U, θ_i die Innentemperatur, θ_a die mittlere Außentemperatur. Die Wärme fließt immer von der höheren zur niederen Temperatur.
Eine kleine Mathematikaufgabe ist die Breite G der Wohnetagendecke. Diese berechnet sich nach der Verhältnisgleichung F / (0,5 B) = (F+D□C) / (0,5 G), oder in Worten: Dachhöhe F verhält sich zur halben Hausbreite wie die Spitzdachhöhe zur halben Deckenbreite. Aufgelöst nach G erhalten wir die Formel

$$G = \frac{(F + D - C)0{,}5B}{0{,}5F} = \frac{(F + D - C)B}{F} = \frac{SB}{F}$$

mit der Spitzdachhöhe S = F+D□C. Die Länge der schrägen Linien H vom Kniestock bis zur Etagendecke (Breite einer der beiden schrägen Dachflächen) berechnet sich nach Pythagoras:

$$H = \sqrt{(C - D)^2 + ((B/2) - (G/2))^2}$$

H ist hier die Hypotenuse eines rechtwinkligen Dreiecks mit den Katheten *Deckenhöhe minus Kniestock* und *halbe Hausbreite minus halber Etagendeckenbreite.*
Die Vorder- und Rückwand (Stirnwände) der Wohnetage bestehen aus einem Rechteck und einem Dreieck, von dem die Spitze fehlt. Die Fläche A_{stirn} ist demnach

$$A_{stirn} = BD + BF/2 - GS/2$$

Wir berechnen in der folgenden Exceltabelle alle Wärmeflüsse nach außen durch die Wände, zum Erdboden, zum Dach und durch die Fenster. Am Anfang stehen die Eingangsdaten.

	A	B	C	D	E	F
1	A	10	Hauslänge			
2	B	6	Hausbreite			
3	C	2,4	Deckenhöhe	Wohnetage		
4	D	0,9	Kniestock			
5	E	2,1	Deckenhöhe	Keller		

6	F	3,5	Dachhöhe	Ab Kniest.		
7	Uw	0,23	U-Wert	Wand		
8	Ud	0,18	U-Wert	Dach		
9	Uft	1,1	U-Wert	Fenst+Türe		
10	Tk	14	Temperatur	Keller		
11	Te	22	Temperatur	Wohnetage		
12	Ta	9	Temperatur	außen		Anzahl
13	Kellerfenst	0,8	Fläche	Kel.-Fenst.		4
14	Fenst klein	1,2	Fl. Wand-	fenst klein		2
15	Fenst klein	1,2	Fl. Dach-	fenster		4
16	Fenst groß	2,0	Fläche	Fenst groß		1
17	Tür	2,1	Fläche			2
18						

Nach den Eingangsdaten folgen die eigentlichen Berechnungen. Die von Excel berechneten Zahlenwerte stehen in Spalte B. Die Formeln, die sich dahinter verbergen, finden Sie in der Spalte *Formeln*.

	A	B	C	D	Formeln
19	Kellerwände	67,2	Wandfläche	Keller	=2*(B2*B5+B1*B5)
20	Kellerboden	60	Bodenfläche	Keller	=B1*B2
21	Kellerfenster	3,2	Fensterfl.	Keller	=F13*B13
22	KWoF	64	Wandfl	ohne Fenst	=B19□B21
23					
24	S	1,05	Höhe	Spitzdach	=B4*B6□B5
25	G	1,8	Breite	Etagendecke	=B24*B2/B6
26	Etagenwände	47,91	Fläche	gerade Wand	=2*(B4*B2+B4*B1+ B2*B6/2-B25*B24/2)
27	H	2,58	Breite einer	Dachfläche	=WURZEL((B3-B4)^2+ ((B2/2)-(B25/2))^2)
28	Dachhaut	69,61	Schrägen +	Decke	=2*B27*B1+B25*B1
29	EwoF	43,51	Etagenwände	ohne Fenster	=B26-F14*B14-F16*B16
30	DhoF	64,81	Dachhaut	ohne Fenster	=B28-F15*B15
31	Qpkeller	163,88	Wärmestrom	Keller	=((B19+B20)*B7+B21*B9)* (B10-B12)
32	Etagenfenster	13,4	Fensterfläche	Etage +Türen	=F14*B14+F15*B15+ F16*B16+F17*B17
33	Qpetage	473,37	Wärmestrom	Etage	=(B29*B7+B30*B8+B32*B9)* (B11-B12)
34	Qgesamt	637,25	Wärmestrom	Gesamt	=B31+B33

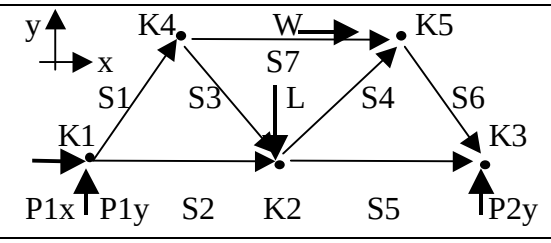
Wir multiplizieren den Wärmestrom Qgesamt mit der Anzahl der Sekunden eines Jahres (365,25*24*3600) und erhalten so die jährlichen Wattsekunden QaWs, eine Energiemenge. Diese rechnen wir in Kilowattstunden um, indem wir durch 1000 und dann durch 3600 teilen und erhalten die Größe QaKWh.

Besorgen Sie sich im Internet den Heizwert H_i von Heizöl (Energieinhalt) in der Form KWh/Kg, die Dichte ρ von Heizöl und den Preis P von einem Liter Heizöl und berechnen Sie mit Excel die Ölmasse M, den Ölverbrauch V in Liter und die jährlichen Ölkosten K in Euro.

Die Formeln sind M = QaKWh / Hi , V = M / ρ , K = V P.

3.6 Zug- und Druckkräfte in einem Stabwerk (Statikberechnung)

Statische Berechnungen werden heutzutage mit ausgefuchsten Programmpaketen gemacht, die die Konstruktion mit den Festigkeitsberechnungen gleich verbinden. Aber zum Üben mit Excel ist die nachfolgende Berechnung gut geeignet. Ein typisches Stabwerk sehen Sie z.B. bei Brückenkonstruktionen, Kranauslegern oder Dachbindern.



Wir haben 5 Knoten **K1, ..., K5** in der x-y-Ebene, an denen Kräfte angreifen, 7 Stäbe **S1, ..., S7** mit willkürlich vorgegebenem Richtungssinn, 2 vertikale Auflagerkräfte **P1y** und **P2y** und eine horizontale Auflagerkraft **P1x**, dazu eine Last **L** und einen Winddruck **W**.

Die Kräfte an einem Knoten müssen im Gleichgewicht sein, sowohl in x- als auch in y-Richtung. Pfeile zum Knoten hin (ankommende Pfeile) zählen wir positiv, Pfeile vom Knoten weg (abgehende Pfeile) zählen wir negativ. Auch diese Regelung ist willkürlich, muss aber einheitlich durchgehalten werden. Natürlich könnte man die Last L auch vom Knoten K2 abgehen lassen, ebenso den Winddruck W vom Knoten K5. Das würde lediglich einen Vorzeichenwechsel bei den berechneten Stabkräften verursachen. Eine Auflagerkraft P2x darf es nicht geben, da dann ein unbestimmtes Gleichungssystem entstünde (eine Brücke wird auf einer Seite befestigt, auf der anderen Seite nur aufgelegt). Für jeden Knoten müssen wir bei einem ebenen Stabwerk zwei Gleichungen aufstellen, eine für die x-Komponenten der Kräfte und eine für die y-Komponenten. Bei räumlichen Stabwerken (z.B. ein Kranausleger) kommt noch eine 3. Gleichung je Knoten für die z-Komponenten hinzu)

Für den Knoten K1 gilt
P1x - S1x - S2x = 0
P1y - S1y - S2y = 0

Für den Knoten K2 gilt
S2x + S3x - S4x - S5x = 0 (Lx ist 0)
(S2y + S3y - S4y - S5y + Ly = 0) bzw. Ly umgestellt
S2y + S3y - S4y - S5y = **–Ly**

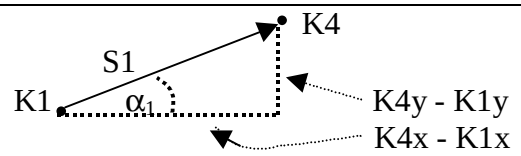
Für den Knoten K3 gilt
S5x + S6x = 0
S5y +S6y + **P2y** = 0

Für den Knoten K4 gilt
S1x - S3x - S7x = 0
S1y - S3y - S7y = 0

Für den Knoten K5 gilt
S4x - S6x + S7x = **–Wx** (Wx schon umgestellt)
S4y - S6y + S7y = 0 (Wy ist 0)

Es entsteht ein Gleichungssystem mit 10 Gleichungen für die 10 Unbekannten **S1, S2, ..., S7, P1x, P1y, P2y**. Die Lasten (bei uns nur eine) und andere Kräfte (bei uns nur Winddruck W) treten als rechte Seite des Gleichungssystem auf. Diese Kräfte sind zahlenmäßig bekannt.

Eine Kraft, z.B. die Stabkraft S1, zerlegen wir nach folgender Formel in ihre zwei Komponenten:
S1x = S1 cos(α1)
S1y = S1 sin (α1).
Das gilt für alle Kräfte. Dabei ist α der Winkel, den die Kraft mit der x-Achse des Koordinatensystems bildet.



Winkel α1 ist die Steigung von S1 und berechnet sich nach der Formel

(1) $$\alpha_1 = \arctan\left(\frac{K4y - K1y}{K4x - K1x}\right)$$

In Excel gibt es eine sehr intelligente Funktion arctan2, die auch Winkel in den negativen x-Quadranten des Einheitskreises berechnen kann. Man muss jedoch Nenner und Zähler des Bruches aus Formel (1) oben als zwei getrennte Argumente einsetzen, d.h.

$$\alpha_1 = \text{arctan2 (K4x –K1x ; K4y – K1y)}$$

Wir gehen demnach folgendermaßen vor:

- Eingabe der Knotennummern 1, 2, 3, 4, 5 in Spalte A
- Eingabe der Knotenkoordinaten x und y in den Spalten B und C
- Eingabe der 7 Stabnummern 1, 2, ..., 7 in Spalte D
- Berechnung der 7 Winkel α_1, α_2, ..., α_7 in Spalte E
- Berechnung der 7 Kosinus- und 7 Sinuswerte zu den 7 Winkeln in den Spalten F und G. Hier "ziehen" wir die Werte, d.h. wir schreiben die Formel nur für α_1.

	A	B	C	D	E	F	G
Z1	Kn	x	y	Stab	Winkel	Kosinus	Sinus
Z2	1	0	0	1	=arctan2(***B5–B2*** ; ***C5–C2***)	=cos(***E2***)	=sin(***E2***)
Z3	2	4	0	2	=arctan2(***B3–B2*** ; ***C3–C2***)		
Z4	3	8	0	3	=arctan2(***B3–B5*** ; ***C3–C5***)		*ziehen*
Z5	4	2	3	4	=arctan2(***B6–B3*** ; ***C6–C3***)	← selektierte Felder	
Z6	5	6	3	5	=arctan2(***B4–B3*** ; ***C4–C3***)	← sind hier fett-	
Z7				6	=arctan2(***B4–B6*** ; ***C4–C6***)	↙ kursiv geschrieben,	
Z8				7	=arctan2(***B6–B5*** ; ***C6–C5***)	z.B. ***B4*** oder ***C6***	

Die Kosinuswerte treten als Koeffizienten der Stabkräfte S1, S2, ..., S7 in den x-Gleichungen auf, die Sinuswerte in den y-Gleichungen. Das Gleichungssystem $\boldsymbol{A\,x = b}$ mit der 10×10-Koeffizientenmatrix $\boldsymbol{A}$, dem 10-Vektor der Unbekannten $\boldsymbol{x}$ und dem 10-Vektor $\boldsymbol{b}$ der rechten Seite lässt sich in älteren Excel-Versionen nur so lösen, dass man zuerst aus der Koeffizientenmatrix $\boldsymbol{A}$ mit der Funktion MINV die inverse Matrix $\boldsymbol{A^{-1}}$ berechnet und dann den Lösungsvektor $\boldsymbol{x}$ nach der Formel $\boldsymbol{x = A^{-1}\,b}$, d.h. mittels Matrix mal Vektor (Funktion MMULT).

Für die Koeffizientenmatrix $\boldsymbol{A}$ benutzen wir die 10×10 Felder ***A10 : J19***, für den Rechte-Seite-Vektor $\boldsymbol{b}$ die Felder ***K10 : K19***. Aus der x-Gleichung für den Knoten K1

$$P1x - S1x - S2x = 0$$

wird

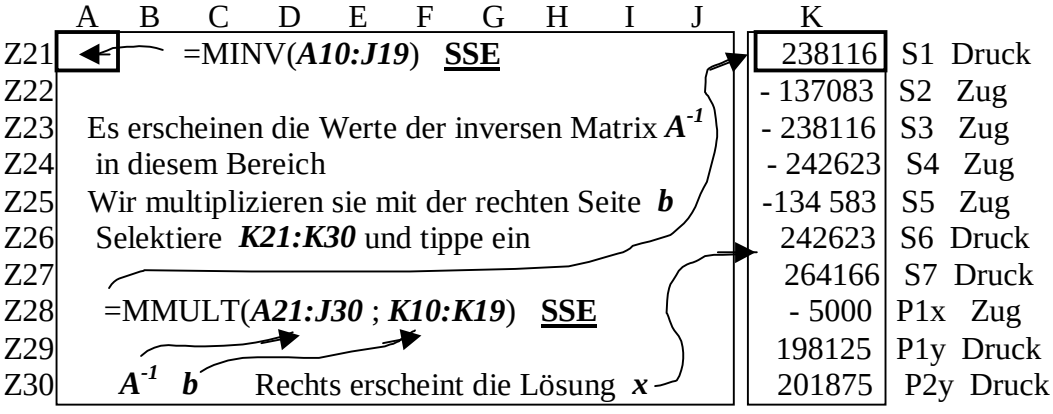
$$-\cos(\alpha_1)\cdot S1 - \cos(\alpha_2)\cdot S2 + 0\cdot S3 + 0\cdot S4 + \ldots\ldots + 0\cdot S7 + 1\cdot P1x + 0\cdot P1y + 0\cdot P2y = 0$$

da wir ja in der Matrix ***A*** jedem Feld einen Wert zuweisen müssen. Die meisten Felder sind Null. Da z.B. der Wert $\cos(\alpha_1)$ auf Feld ***F2*** steht, muss Matrixelement A_{11} (1. Gleichung, 1. Unbekannte) durch folgende kleine Formel berechnet werden: $A_{11} = -\cos(\alpha_1) =$ ***–F2*** . Die vollständige Matrix mit rechter Seite lautet dann:

	A	B	C	D	E	F	G	H	I	J	K	L
Z9	S1	S2	S3	S4	S5	S6	S7	P1x	P1y	P2y	***b***	Bem.
Z10	=–***F2***	=–***F3***	0	0	0	0	0	**1**	0	0	0	
Z11	=–***G2***	=–***G3***	0	0	0	0	0	0	**1**	0	0	
Z12	0	=***F3***	=***F4***	=–***F5***	=–***F6***	0	0	0	0	0	0	
Z13	0	=***G3***	=***G4***	=–***G5***	=–***G6***	0	0	0	0	0	**400000**	Last
Z14	0	0	0	0	=***F6***	=***F7***	0	0	0	0	0	
Z15	0	0	0	0	=***G6***	=***G7***	0	0	0	**1**	0	
Z16	=***F2***	0	=–***F4***	0	0	0	=–***F8***	0	0	0	0	
Z17	=***G2***	0	=–***G4***	0	0	0	=–***G8***	0	0	0	0	
Z18	0	0	0	=***F5***	0	=–***F7***	=***F8***	0	0	0	**-5000**	Wind
Z19	0	0	0	=***G5***	0	=–***G7***	=***G8***	0	0	0	0	

Erklärung der rechten Seite: Da die Last von etwa 40 t = 400.000 N als –Ly in der Knotengleichung von Knoten K2 steht, aber der Pfeil in die negative y-Richtung zeigt, geben wir sie positiv ein wegen –(–400000) = 400000. Da der Winddruck in der Knotengleichung für K5 als –Wx steht, der Pfeil in die positive x-Richtung zeigt, geben wir den Winddruck negativ ein, d.h. +(–5000) = –5000. Die Zahlen 400000 und 5000 sind ein Rechenbeispiel.

Aus der Koeffizientenmatrix ***A*** erzeugen wir mit der Funktion MINV die inverse Matrix $\boldsymbol{A^{-1}}$ und dann den Lösungsvektor ***x*** nach der Formel $\boldsymbol{x = A^{-1}\ b}$, d.h. mittels Matrix mal Vektor (Funktion MMULT, eigentlich Matrixmultiplikation). Selektieren Sie die Felder ***A21:J30*** als Platz für die Matrix $\boldsymbol{A^{-1}}$ und tippen Sie ein (**SSE** ist Strg-Shift-Enter):



Die Stäbe 1, 6 7 werden auf Druck belastet. Die Stäbe 2, 3, 4, 5 werden auf Zug belastet. Lagerkraft P1x fängt den seitlichen Winddruck auf (Zugkraft), Lagerkräfte P1y und P2y (Druckkräfte) tragen die Last, durch den Winddruck etwas ungleichmäßig belastet.

3.7 Beispiel Hefewachstum

Futterhefe wird aus Hefezellen gewonnen, die in Fermentern wachsen. Ein Substrat (z.B. Glukose mit einigen Mineralzusätzen) wird mit Wasser und einer Startmenge Frischhefe versetzt und mit reichlich Zufuhr an gefilterter Luft zum Wachsen gebracht. Ist zu wenig Sauerstoff verfügbar, dann erzeugen die Hefezellen Alkohol und wir sprechen von Gärung. Nach einigen Tagen (etwa 5-14 je nach Temperatur, Hefe und Feed) kann man die Zellen von der Flüssigkeit trennen, trocknen, mahlen und verpacken.

Den Fermentationsprozess versucht man durch ein Wachstumsmodell zu beschreiben. Dieses Modell erlaubt eine bessere Prozesssteuerung. Wachstumsmodelle sind Systeme von gekoppelten Differentialgleichungen (Mathematik 2). Man kann sie auf simple Weise numerisch lösen, z.B. nach dem Verfahren von Euler-Cauchy.

Beispiel einer einfachen Wachstumskurve: Ein Kilo suspendierte Hefe vermehrt sich unter guten Bedingungen mit einer Rate (Wachstumskoeffizient μ) von etwa $\mu = 0{,}03$ [1/h], d.h. nach einer Stunde sind aus einem Kilo Hefe 1,03 Kilo geworden (ein Kilo Ausgangsmasse und 0,03 Kg Zuwachs), nach zwei Stunden 1*1,03*1,03=1,061 Kilo usw. Bezeichnen wir die Hefemenge selbst mit y, den Zuwachs mit dy, die Zeiteinheit (z.B. eine Stunde) mit dt, dann erhalten wir die Gleichung

$$\frac{dy}{dt} = \mu\, y$$

Das ist eine einfache Differentialgleichung. Die exakte Lösung ist die Exponentialfunktion $y(t) = Y_o\, e^{\mu t}$ .Dabei ist Y_o die Startmasse an Hefe. Wir bringen das dt auf die rechte Seite der Differentialgleichung und erhalten $dy = \mu\, y\, dt$. Wir haben damit eine simple Gleichung gefunden, wie man zu einer gegebenen Hefemenge y den jeweiligen Zuwachs dy berechnen kann.

Das numerische Verfahren von Euler ist rechts erklärt. Es liefert mit ausreichender Genauigkeit das Anwachsen der Größe y von Zeitschritt zu Zeitschritt in Form einer Exponentialkurve, aber nicht als Formel, sondern als Zahlenreihe. Diese können wir grafisch als Kurve darstellen.	Startmenge ist y=Yo, Startzeit to=0 Zuwachs ist → $dy = \mu\, y\, dt$ Neues y ist $y_{neu} = y_{alt} + dy$ Neue Zeit ist $t_{neu} = t_{alt} + dt$ Wir setzen $y_{alt} = y_{neu}$ $t_{alt} = t_{neu}$ und wiederholen

Beim Wachsen der Hefe wird Substrat verbraucht. Einmal entsteht neue Hefemasse, andererseits verbrauchen die Lebensprozesse der Hefe Energie. Als Abprodukt entsteht in erster Linie Kohlendioxid (CO_2), daneben aber eine Reihe weiterer Substanzen. Was uns interessiert: wie viele Kilo Substrat benötigen wir, um ein Kilo Frischhefe zu produzieren. Wir nennen diese Größe hier den Substrateinsatzkoeffizienten k [Kg Substrat /Kg Hefe]. Wir nehmen k = 3,5 an, d.h. aus 3,5 Kg Substrat erzeugen wir 1 Kg Frischhefe.

Eine Fermentationsstrategie ist z.B. der Batch. Der Fermenter wird am Beginn mit einer gewissen Substratmenge S_o gefüllt. Aus dem Behältervolumen V und der gerade vorhandenen Substratmenge S berechnet man die Substratkonzentration c_s [Kg/l] = S / V. Beim Wachsen

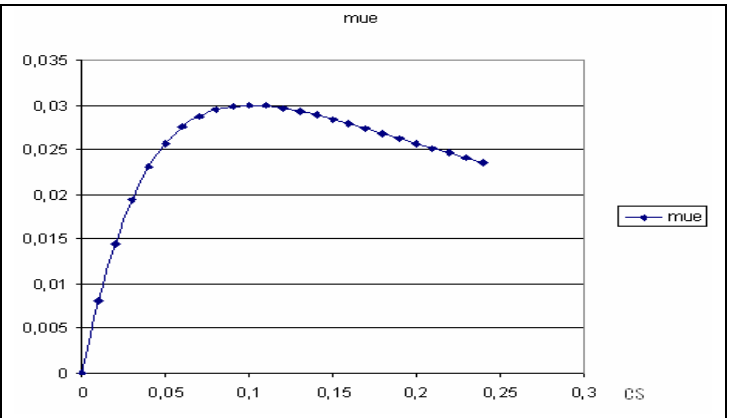
der Hefe wird das Substrat verbraucht. Die Substratmenge S und damit die Substratkonzentration c_s nehmen ab. Ist kein Substrat mehr vorhanden, dann hört das Wachstum auf. Die Differentialgleichung für die Abnahme des Substrats ist dann:

$$dS = -k\,dy = -k\,\mu\,y\,dt \quad bzw. \quad S_{neu} = S_{alt} + dS \quad oder \quad S_{neu} = S_{alt} - k\,\mu\,y\,dt$$

Kinetik heißt die Lehre von der Geschwindigkeit, mit der Prozesse ablaufen. Das Wachstum μ hängt u.a. auch von der Substratkonzentration c_s ab, d.h. μ ist eine Funktion von c_s . Eine bekannte Kinetikformel stammt von Haldane:

$$\mu(c_S) = \frac{\mu^* c_S}{K_S + c_S + K_I\,c_S^2}$$

Unser Wachstum hat hier im Bild ein Maximum bei c_s=0,1 oder bei 10% Substratkonzentration.



Die Haldane-Kinetik verwendet hier drei Konstanten: μ*, K_S und K_I. Diese werden aus Fermenterversuchen durch Kurvenfit so bestimmt, dass die Kurve $\mu(c_s)$ sich bestmöglich den gemessenen Wachstumswerten bei unterschiedlichen Substratkonzentrationen c_s anpasst. Die Konstante μ* legt zusammen mit K_S das theoretische Maximalwachstum fest. Die Konstante K_S legt fest, wie schnell der anfänglich gerade Anstieg von $\mu(c_s)$ abknickt und für $c_s \to \infty$ in eine waagrechte Gerade in der Höhe μ* einmünden würde. Dass die Kurve wieder absinkt, wird von dem Term $K_I\,c_s^2$ im Nenner der Haldane-Formel bewirkt. Wir verwenden in unserer Wachstumssimulation folgende Zahlen:

$$\mu^* = 0{,}09 \quad , \quad K_S = 0{,}10 \quad \text{und} \quad K_I = 10.$$

Mortalität: Hefezellen leben nicht ewig. Wenn sie etwa zehnmal gesprosst haben, dann sterben sie ab. Sie sterben aber auch ab, wenn längere Zeit kein Substrat verfügbar ist oder ihr Lebensraum zu beengt oder vergiftet wird. Um die Mortalität M zu modellieren, berechnen wir die Hefekonzentration c_H = y/V in [Kg/l]. Wir machen den simplen Ansatz, dass mit steigender Hefekonzentration die Mortalität zunimmt und einen Teil des Wachstums frisst. Die toten Hefezellen sind jedoch nicht nutzlos, sondern Teil der Ernte. Das Wachstum μ hängt demnach nicht mehr nur von der Substratkonzentrationen c_s ab, sondern auch von der Hefekonzentration c_H. Wir machen hier einen denkbar simplen Ansatz: Das Wachstum sinkt linear, wenn sich die Hefekonzentration der kritischen Konzentration c_{HMax} nähert.

$$\mu(c_S, c_H) = \mu(c_S)\,M(c_H) \quad \text{mit} \quad M(c_H) = (c_{HMax} - c_H)/c_{HMax}$$

Je weiter sich demnach die Hefekonzentration c_H dem Maximalwert c_{HMax} nähert, desto mehr wird das Wachstum durch eine erhöhte Mortalität eingeschränkt. Natürlich sind auch andere Modellformeln möglich. Für c_{HMax} legen wir den Wert c_{HMax} = 0,1 fest, d.h. wenn ein Anteil an 10% Frischhefe im Fermenter vorliegt, hört das Wachstum auf.

Zusammenfassung der Daten und Formeln:

S_o = 1000 Kg Glukose als Startmenge an Trockensubstrat
Y_o = 50 Kg Frischhefe als Startmenge für die Hefe
dt = 1 Stunde als Zeitschritt

V = 10 m^3 oder 10.000 Liter als Flüssigkeitsvolumen des Fermenters
$\mu^* = 0{,}09$, $K_S = 0{,}10$ und $K_I = 10$ als Parameter der Haldane-Kinetik
$c_{HMax} = 0{,}1$ für die maximale Hefekonzentration in der Mortalitätsformel
k = 3,5 als Substrateinsatzkoeffizient

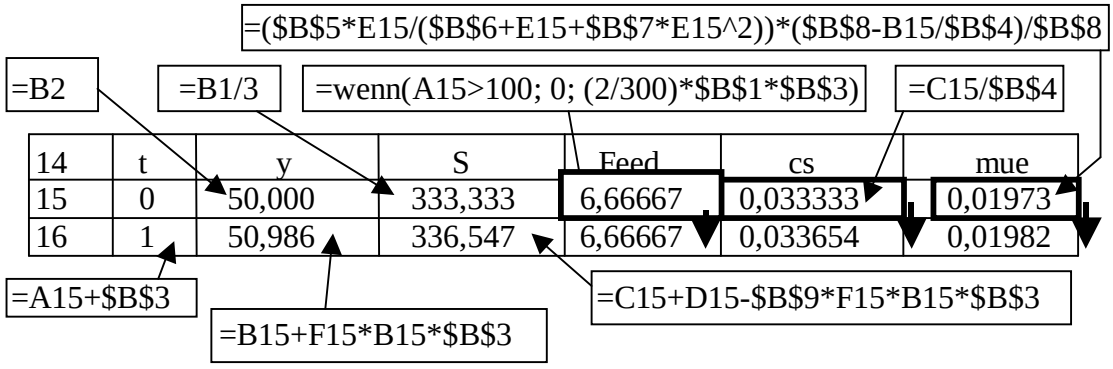
dy = μ y dt als Hefezuwachs pro Zeiteinheit
dS = □ k μ y dt oder dS = □ k dy als Substratabnahme pro Zeiteinheit
$\mu(c_S, c_H) = \mu(c_S) M(c_H)$ als Wachstumskoeffizient mit der Haldane-Kinetik

$\mu(c_S) = \frac{\mu^* c_S}{K_S + c_S + K_I c_S^2}$ und mit $M(c_H) = (c_{HMax} - c_H)/c_{HMax}$ als Mortalitätsterm.

Bei der Programmierung der Exceltabelle muss man aufpassen, dass man keine Zirkelbezüge programmiert. Die Tabelle beginnt wie immer mit den Eingangsdaten.

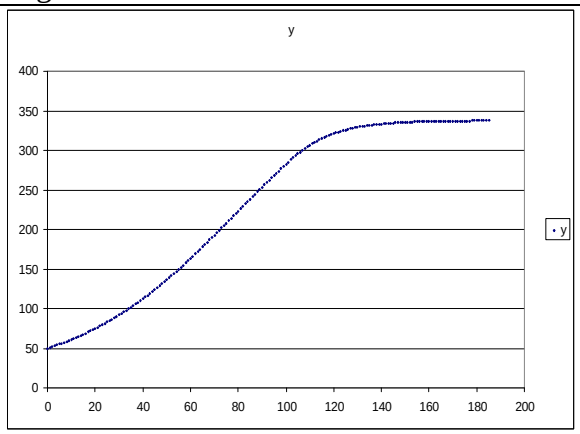
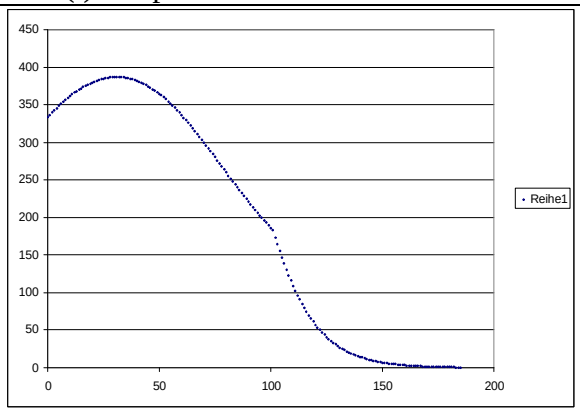
Zeile	A	B	C	D	E
1	So	1000	Startmenge	Substrat	Kg
2	Yo	50	Startmenge	Hefe	Kg
3	dt	1	Zeitschritt	in Stunden	h
4	V	10000	Volumen	Flüssigkeit	Liter
5	mue_stern	0,09	Konstante	Haldane	Kinetik
6	Ks	0,1	Konstante	Haldane	Kinetik
7	Ki	10	Konstante	Haldane	Kinetik
8	cHMax	0,1	Konstante	Mortalität	
9	k	3,5	Substrat-	einsatzkoeff.	

Für die Berechnung der Kurven aus den Differenzengleichungen (das sind vergröberte Differentialgleichungen) sind nur die ersten beiden Zeilen der Tabelle wichtig. Der ganze Rest der Tabelle ergibt sich dann durch Ziehen. Die Formeln hinter den berechneten Zahlen sind in extra Boxen aufgelistet. Auf die Konstanten (Eingangsdaten) in Zellen, die später gezogen werden, muss mit festen Bezügen ($-Zeichen verwenden) zugegriffen werden.



- Die Schritte in diesem Tabellenteil sind:
- Sie tippen in Zelle A15 eine Null ein als Startwert für die Zeit t
- Sie kopieren mit der Anweisung =B2 die Startmenge der Hefe nach Zelle B15
- Sie geben mit =B1/3 eine Startmenge von $S_o/3$ für das Substrat S vor in Zelle C15
- Sie berechnen in Zelle D15 nach der Formel *=wenn(t>100; 0; (2/300)*S_o*dt)* die Substratdosierung (Feed) für die nächsten 100 Stunden (pro Stunde 1/100 von 2/3 von S_o.) In Excelnotation =wenn(A15>100; 0; (2/300)*$B$1*$B$3)

- Sie berechnen in Zelle E15 mit =C15/$B$4 die Substratkonzentration cs = S/V
- Sie berechnen mit =($B$5*E15/($B$6+E15+$B$7*E15^2))*($B$8-B15/$B$4)/$B$8 auf Zelle F15 den Wachstumskoeffizienten (Haldane-Kinetik und Mortalität) nach der Formel $\mu = (\mu^* c_s / (K_s + c_s + K_i c_s^2)) (c_{HMax} - y/V) / c_{HMax}$
- Sie berechnen in Zelle A16 die neue Zeit als $t_{neu} = t_{alt} + dt$ (In Excel =A15+$B$3)
- Sie berechnen in Zelle B16 die angewachsene Hefemenge (=B15+F15*B15*$B$3) nach der Formel $y_{neu} = y_{alt} + \mu\ y\ dt$. Wichtig ist dabei der Bezug auf das μ in der vorangehenden Zeile, da dieses μ bereits berechnet ist.
- Sie berechnen in Zelle C16 das neue S (=C15+D15-$B$9*F15*B15*$B$3) nach der Formel $S_{neu} = S_{alt}$ + Feed □ k μ y dt. Auch hier beziehen wir uns auf das μ der vorangehenden Zeile.
- Sie ziehen die Zellen D15, E15 und F15 in die jeweils darunter liegende Zelle
- Sie selektieren die Zellen A16:F16 und ziehen diese Zeile immer weiter nach unten bis Sie sehen, dass die Hefemenge y in Spalte B nur noch unwesentlich wächst (Bei dt=1 etwa bis Zeile 200)

Sie selektieren die Zeitspalte t und die daneben liegende Spalte y mit den berechneten Zahlenwerten → Einfügen Diagramme → Diagrammtyp xy → Fertigstellen. Das Ergebnis ist die Kurve y(t), die die Zunahme der Hefemenge mit der Zeit zeigt.	Sie kopieren mit der Anweisung =A14 das t in Spalte H14. Sie kopieren mit der Anweisung =C14 das S in Spalte I14. Sie selektieren die beiden Zellen H14:I14 und ziehen. Sie erhalten eine Kopie der t- und der S-Spalte nebeneinander. Damit erzeugen Sie die S(t)-Graphik wie unten.
	

Der Zeitschritt dt=1 ist für eine etwas exaktere Berechnung eigentlich zu grob. Versuchen Sie es auch mit dt=0,25 oder dt=0,1. Natürlich müssen Sie dann die Tabelle deutlich länger ziehen, z.B. bis etwa Zeile 800 im Falle dt=0,25 bzw. bis etwa Zeile 2000 im Falle dt=0,1.

Der Knick in der S(t)-Kurve rührt übrigens daher, dass die Substratdosierung in diesem Beispiel bei t =100 Stunden endet. Wollen Sie nach einem Dosierungsplan vorgehen, der sich schlecht programmieren lässt, dann ziehen Sie in der Spalte Feed eine Null von Anfang bis Ende durch und setzen anschließend per Handarbeit die einzelnen Dosierungen zur richtigen Zeit t ein. Wichtig ist nur, dass die Gesamtmenge S_o eingehalten wird.

3.8 Einfache lineare Regression mit Teststatistiken in Excel

Die Funktion **=trend(y-Werte ; x-Werte)** berechnet die Erwartungswerte $\hat{y}_i$ der Ausgleichsgeraden, die durch die y- und x-Werte definiert ist. Die Funktion

=rgp(y-Werte ; x-Werte ; wahr ; wahr) berechnet die Regressionskoeffizienten, deren Standardabweichungen, die Reststreuung, die Bestimmtheit r^2, deren Freiheitsgrad usw. einer einfachen oder multiplen Regression. Das erste *wahr* steht für ein "Modell mit Regressionskonstante", das zweite *wahr* für "außer den Koeffizienten weitere statistische Kennzahlen ausgeben", wie oben genannt. Die Abkürzung **SSE** steht im Schema unten für die 3-fach-Tastenbelegung Strg-Shift-Enter. Drücken Sie erst die beiden linken Tasten Strg und ⇑, dann zusätzlich ENTER. Zuerst tippen Sie die Spaltenbezeichnungen x, y, y-Dach als Text ein, dann die y-Zahlenwerte in die Felder A2 bis A7, dann die x-Zahlenwerte in B2 bis B7, dann laut Schema:

	S1=A	S2=B	S3=C
Z1	x	y	y-Dach
Z2	1,7	3,3	
Z3	2,3	4,1	
Z4	2,1	4,5	
Z5	2,4	4,7	
Z6	3,9	8,3	
Z7	1,6	3,3	
Z8			
Z9	2,20	-0,45	b1,bo
Z10	0,18	0,45	sb1,sbo
Z11	0,97	0,34	r2, sR
Z12	144	4	F, df
Z13	16,8	0,47	ssreg,ssres
Z14			
Z15	12	-1.0	t1, t2

Selektiere *C2:C7* und tippe ein:
=trend(*sel. B2:B7* **;** *sel. A2:A7* **)** **SSE**
(y-Werte x-Werte)
Auf *C2 bis C7* erscheinen die berechneten y-Dach-Werte. Jetzt wollen wir die Koeffizienten und Statistiken berechnen: Selektiere *A9:B13* und tippe ein:
=rgp(*sel.B2:B7***;***sel.A2:A7***;wahr;wahr)** **SSE**
Es erscheinen die Zahlen in Spalte A und B z.B. b1=Anstieg, bo=Regressionskonstante der Geraden y = bo + b1 x
Berechnung der t-Statistiken: Sel. *A15:B15*
= *sel.A9:B9* / *sel. A10:B10* **SSE**
die beiden Teststatistiken erscheinen

In den berechneten Statistiken bedeuten:

sb1, sb0 die Standardabweichungen (Fehler) der beiden Koeffizienten b_1 und bo

r^2 die multiple Bestimmtheit (bei einer einfach linearen Regression ist es das Quadrat des Korrelationskoeffizienten r)

sR Reststreuung der Messpunkte um die Gerade (mittlere Abweichung)

F Testgröße (F-Statistik) hier zur Hypothese Ho: $b_1=0$ mit den Freiheitsgraden df1=1 und df2=df. Bei einer einfachen Regression wie hier im Beispiel ist $F=t_1^2$, und t_1 die t-Statistik für b_1 mit df Freiheitsgraden.

t_1, t_2 sind die Teststatistiken zu den Koeffizienten b1 und b0. Man testet damit die Hypothesen Ho: b1=0 bzw. Ho: bo=0

ssreg $= \sum (y_i - \bar{y})^2$, auch Summe der Abweichungsquadrate der y genannt (SAQyy)

ssresid $= \sum (\hat{y}_i - y_i)^2 = \sum e_i^2$, auch Summe der Abweichungsquadrate bzw. Fehlerquadratsumme genannt.

Wir wollen eine Graphik der Regressionsgeraden mit den Messpunkten als gif-Datei in ein WORD-Dokument einfügen:

1. Selektiere *A1:C7*, d.h. die Spaltenbezeichner werden hier mitselektiert.
2. Klicke auf den Diagrammassistenten (das Ikon mit dem kleinen Säulendiagramm)
3. Wähle Diagrammtyp *Punkte (X,Y)* und dazu die Darstellung *"Linien mit Punkten"*
4. Klicke auf *Ende* und dann auf die Graphik (Sie muss jetzt schwarz umrandet sein). Klicke auf *Bearbeiten*, dann auf *Kopieren*, dann Excel minimieren (Das Minuszeichen ganz oben rechts)
5. ---> Alle Programme ---> Zubehör ---> Paint ---> Koffersymbol (Einfügen) ---> Speichern unter ---> Dateityp *.gif, Dateiname *Trend*, Ordner IV. Paint schließen.

6. ---> MS-Word starten ---> Datei ---> Neu ---> 3 Leerzeilen (3 mal ENTER) ---> Einfügen ---> Graphik ---> von Datei ---> wählen Sie aus Ihrem IV-Ordner die gerade gespeicherte Datei *Trend.gif*. Das WORD-Dokument speichern unter Trend in ihrem IV-Ordner. WORD schließen und Excel wieder maximieren.

3.9 Multiple Regression mit Excel

Die multiple Regression schätzt aus p Einflussgrößen X_1, X_2, …, X_p die Werte einer Zielgröße Y. Das am meisten benutzte Regressionsmodell ist die Ebenengleichung

$$Y_i = b_0 + b_1 X_{i1} + b_2 X_{i2} + ... + b_p X_{ip} + e_i$$

Dabei ist Y_i ein beobachteter Wert der Zielgröße, X_{ij} ist der i-te Wert der j-ten Einflussgröße, b_0 ist die Regressionskonstante, b_1 , b_2 , ... , b_p sind Regressionskoeffizienten, e_i ist der Fehler im Datenpunkt i (oder Abweichung bzw. Residuum).

- Die Multiple Regression kann Zielgrößenwerte vorhersagen (Prognose, Vorhersage).
- Man kann den Gesamteinfluss aller Einflussgrößen auf die Zielgröße global bewerten.
- Man kann wesentliche Einflussgrößen erkennen, indem man sie einzeln bewertet.

Beispiel Pflanzenwachstum: Der Ertrag in Abhängigkeit unterschiedlicher Parameter wird bestimmt. Die verfügbaren Daten sind in der folgenden Exceltabelle zu sehen.

Zeile	A	B	C	D	E	F
1	Bodenwert	Beregnung	Düngung	Temperatur	Bodendichte	Y = Ertrag
2	2	2	0,10	17	1320	1,1
3	2	3	0,15	19	1410	1,5
4	4	2	0,10	22	1190	1,8
5	3	4	0,20	20	1240	2,0
6	2	1	0	18	1240	0,80
7	1	3	0,10	18	135	1,20
8	4	4	0	21	127	1,95
9	2	3	0,20	15	1300	1,15

Wir markieren das Feld von A11:F15 und tippen eine Regressionsanweisung ein, die zuerst die Zielgrößenwerte Y nennt, dann die Einflussgrößenwerte X. Das erste „*wahr*“ legt ein Modell „*mit Konstante*“ fest, das zweite „*wahr*“ legt fest, dass wir außer den Koeffizienten weitere Werte berechnet haben möchten, z.B. die s_{bi}, R^2, s_R, usw. Es sind immer 5 Zeilen, die Sie markieren. Die Spaltenzahl richtet sich jedoch nach der Anzahl der Koeffizienten im Regressionsmodell (bo zählt mit, falls es berechnet werden soll).

=rgp(F2:F9; A2:E9; wahr; wahr) Strg-Shift-Enter

Zeile 11	b_5=0,000129	b_4=0,0995	b_3=1,379	b_2=0,185	b_1=0,137	b_0= -1,597
12	s_{b5}=0,000369	s_{b4}=0,0119	s_{b3}=0,253	s_{b2}=0,0214	s_{b1}=0,0322	s_{b0}=0,514
13	R^2=0,997	s_R =0,0431				
14	F= 147,44	FG= 2				
15	ssreg=1,37	ssresid=0,0037	ssreg	ssresid		
16	Dichte	Temperatur	Düngung	Beregnung	Bodenwert	b_0

Wie man sieht, kehrt Excel die Reihenfolge der Regressionskoeffizienten um (b_5, b_4, ..., b_0). In den berechneten Statistiken in den Zeilen 12 bis 15 bedeuten im Falle der multiplen Regression:

s_{b5},..., s_{b0}	die geschätzten Standardfehler der Koeffizienten b_5 ,..., b_o
R^2	die multiple Bestimmtheit (bei einer einfach linearen Regression ist es das Quadrat des Korrelationskoeffizienten r. R^2 =0 heißt, dass keinerlei linearer Zusammenhang zwischen der Gesamtheit aller Einflussgrößen mit der Zielgröße besteht. R^2 =1 heißt, dass die Einflussgrößen die gegebenen y-Werte absolut exakt reproduzieren ohne jede Abweichung.)
s_R	Reststreuung der Messpunkte um die berechnete Ebene (mittlere Abweichung)
F	Testgröße (F-Statistik nach Fisher) zur Bewertung der multiplen Bestimmtheit. Hypothese Ho: „keine Bestimmtheit, ein Wert von R^2>0 ist rein zufällig“. Hypothese Ha: „Es besteht ein signifikanter Einfluss der Einflussgrößen auf die Zielvariable, ein Wert von R^2>0 ist nicht zufällig“. Die Irrtumswahrscheinlichkeit p bei Ablehnung von Ho (bzw. Annahme von Ha) berechnet man mit der Funktion FVERT(F ; n – FG – 1 ; FG) wenn b_o mitberechnet wird (mit TRUE ausgewählt), und wird auch als p-Value zum F-Test bezeichnet. Falls b_o nicht berechnet wird (mit FALSE abgewählt), schreiben Sie FVERT(F ; n – FG ; FG).

Die Summen ssreg und ssresid wurden schon bei der einfachen Regression kurz beschrieben. Für die Bewertung der Wichtigkeit der einzelnen Einflussgrößen bzw. der Konstanten für das Regressionsmodell hat man zu jedem Koeffizienten das Hypothesenpaar Ho und Ha. Ho sagt: „Diese Einflussgröße hat keinen linearen Einfluss auf die Zielgröße. Ein Wert bj ≠ 0 eines Koeffizienten ist rein zufällig“. Hypothese Ha sagt: „Diese Einflussgroße trägt signifikant zur Erklärung der Zielgröße bei.“ Praktisch berechnet man zu jedem Koeffizienten eine Teststatistik. Meistens wird die t-Statistik verwendet. Es gilt $t_i = |b_i / s_{bi}|$. Wir dividieren mit einer Excelanweisung gleich alle Koeffizienten und die Konstante durch ihren geschätzten Standardfehler und bilden den Absolutbetrag. Dazu markieren wir die Felder A18:F18 und tippen die nachfolgende Befehlszeile ein:

=ABS(A11:F11/A12:F12) Strg-Shift-Enter

Zeile 18	t_5=0,351	t_4=8,33	t_3=5,43	t_2=8,65	t_1=4,27	t_0=3,10

Die t-Verteilung hat eine ähnliche Gestalt wie die Normalverteilung (Glockenkurve). Die Funktion TVERT berechnet aus einem t-Wert, dem Freiheitsgrad FG von oben und der Zahl 2 die zweiseitige Irrtumswahrscheinlichkeit (p-Value) bei Ablehnung der Hypothese H_0 zum betreffenden Koeffizienten. Dieser p-Value (die Irrtumswahrscheinlichkeit) sollte möglichst klein sein, z.B. <0,05, denn dann bewerten wir die Einflussgröße als wesentlich (signifikant). Zur Berechnung der p-Values markieren wir die Zellen A20:F20 und tippen folgende Anweisung ein:

=TVERT(A18:F18; B14; 2) Strg-Shift-Enter

Zeile 20	p_5 = 0,75	p_4 = 0,014	p_3 = 0,032	p_2 = 0,013	p_1 = 0,0506	p_0 = 0,09

In der Forschung gibt man meist eine zulässige Irrtumswahrscheinlichkeit α = 5% (0,05) vor. D.h. mit 5% Wahrscheinlichkeit wollen wir uns bei der Bewertung einer Einflussgröße irren dürfen. Ist der berechnete p-Value größer α, dann entscheiden wir uns für Hypothese Ho (unwesentliche Einflussgröße). Ist $p \leq \alpha$, dann entscheiden wir uns für Hypothese Ha (wesentliche Einflussgröße). Zur Darstellung der Hypothesenwahl markieren wir die Felder A22:F22 und tippen folgende Anweisung ein:

=wenn(A20:F20 > 0,05 ; „Ho“ ; „ Ha“) Strg-Shift-Enter

Zelle 22	Ho	Ha	Ha	Ha	Ho	Ho
23	Dichte	Temperatur	Düngung	Beregnung	Bodenwert	b_0

Den schlechtesten p-Value (höchste Irrtumswahrscheinlichkeit) hat Einflussgröße X_5=Dichte. Wenn wir unser Regressionsmodell von unwesentlichen Bestandteilen befreien wollen, sollten wir zuerst diese Einflussgröße entfernen (Schrittweiser Abbau). Entfernen Sie jedoch in jedem Schritt immer nur einen Term, d.h. eine Einflussgröße oder die Konstante bo. Durch Korrelationen zwischen den Einflussgrößen ändern sich die p-Values oft dramatisch bei Wegnahme oder Hinzunahme einer einzelnen Einflussgröße. Die Regressionskonstante bo kann man entfernen, indem man statt des ersten „*wahr*“ in der rgp-Anweisung ein „*falsch*“ schreibt.

Den globalen Test auf einen signifikanten linearen Zusammenhang der Gesamtheit der Einflussgrößen auf die Zielgröße macht man mit dem F-Test (siehe oben bei der Erklärung des F). Das folgende Rechenschema liefert den p-Value und die Hypothese Ho bzw. Ha. Jede eingetippte Anweisung schließen Sie mit ENTER ab.

	A	B	C	D	E	F
	=ANZAHL(A2:A9)		=FVERT(A14; A24-B14-1; B14)			=wenn(….
Zeile 24	**8**		**0,0067**			**Ha**
Zeile 25	Anzahl		p-Value			Hypothese

Die Wenn-Anweisung, die Sie in Zelle F24 eintippen, lautet vollständig:

=wenn(C24 > 0,05 ; „Ho“ ; „Ha“)

3.10. WinSTAT Statistiksoftware (nur informativ)

Dieser Teil des Skripts ist rein informativ. Es gibt einige frei verfügbare oder preisgünstige Statistikprogramme auf dem Markt bzw. im Web, die über die Leistung von Excel hinausgehen und bei aufwendigeren statistischen Untersuchungen verwendet werden können. Eines dieser Programme ist z.B. WinSTAT (Einzellizenz ca. 90 €, Klassenraumlizenz ca. 350 €) WinSTAT ist ein Add-In zu EXCEL von Robert K. Finch. Man bleibt in EXCEL, d.h. die Daten werden als EXCEL-Tabellen verwaltet und benutzt. Die Ergebnisse fallen als EXCEL-Tabellen an. WinSTAT kann:

1. Daten kontrollieren, transformieren (Mittelwerte, Quantile, Boxplots, Histogramme, ...)
3. Daten analysieren (Mittelwertvergleiche von 2 bzw. N Gruppen, Korrelationen, Regressionen, Diskriminanz-, Cluster-, Faktorenanalyse, Survivalanalyse)

Statistik mit WinSTAT

Starten EXCEL mit WinSTAT, Datei laden, statistische Maßzahlen.
WinSTAT Add-In von Robert K. Finch erkennen Sie an der zweiten Menüleiste mit den Buttons:

Statistik: Grundlagen, Mittelwertvergleiche, Korrelation, Regression, Diskriminanz-, Cluster-, Faktoren-, Survivalanalyse und Prozessfähigkeit
Graphik: Histogramm, Mittelwerte, Box&Whiskers, Streudiagramm, kumulative Häufigkeit, Qualitätssicherung, Pareto-Diagramm
Daten: Zeilen ein- und ausblenden, Datenbankbereich festlegen

Hilfe:	Das WinSTAT-Handbuch als Themenkomplexe aufgegliedert
←	Zum Datenbankbereich (geht schneller mit der Leiste ganz unten)
X	Ergebnisblatt löschen (braucht man ständig, wenn man herumprobiert)

Die beiden letzten Buttons benötigen wir erst einmal nicht. Der Rest dieser Menüleiste ist die vom originalen EXCEL

Wir wollen die Datei **Kinder.xls** laden: →Datei →Öffnen → System C →Programme →WinSTAT →Beispiele →Kinder.xls öffnen
→Daten →Datenbankbereich setzen →"alle verwendeten Zellen" →OK
→Statistik →Grundlagen →Deskriptiv →Merkmal ***Größe*** anklicken →OK
Es erscheint eine lange Spalte mit verschiedenen statistischen Maßzahlen

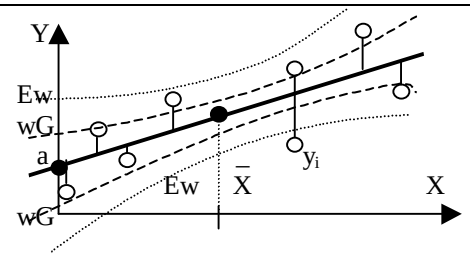
Filter in EXCEL: z.B. Mittelwerte des Merkmals Größe in Altersklassen bestimmen nur für Jungen.
Markiere Spalte C →Daten(aber oberste Menüleiste) →Filter →Autofilter →PopUp-Pfeil in Spalte C anklicken →***Junge*** anklicken. Alle Zeilen mit *Mädchen* werden ausgeblendet, bis wir den Filter rückgängig machen.

Einfache Regression: Datei Autos.xls laden. Wir legen in Spalte P eine neue Variable *Residuen* an und füllen sie mit Nullen (eine 0 und dann ziehen). Datenbankbereich festlegen. Wir probieren eine einfache Regression: →Statistik →Regression →einfache → x-Variable: *PS* anklicken, y-Variable: *Verbrauch*, *Residuen schreiben* anklicken, Variable *Residuen* anklicken, OK
Es erscheinen die Koeffizienten, die Korrelation und die Graphik mit den Konfidenzkorridoren.

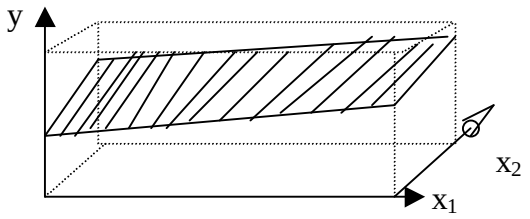
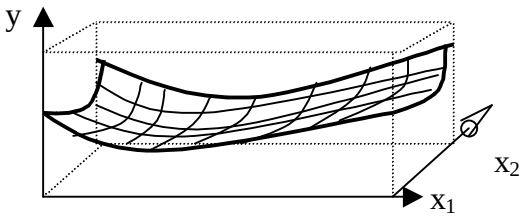
$$\hat{y}_i \pm t(\alpha, FG, zweis.) \cdot \hat{S}_R \cdot \sqrt{\frac{1}{n} + \frac{(x_i - \bar{x})^2}{SAQxx}}$$ Konfidenzintervall der wahren Regressionsgeraden

$$\hat{y}_i \pm t(\alpha, FG, zweis.) \cdot \hat{S}_R \cdot \sqrt{1 + \frac{1}{n} + \frac{(x_i - \bar{x})^2}{SAQxx}}$$ Konfidenzintervall der Einzelwerte bei Prognose.

Zieht man immer wieder neue Stichproben des Umfangs n und berechnet man aus jeder Stichprobe die Regressionsgerade, dann erwartet man 100-α% der Geraden im Konfidenzintervall der "**wahren Geraden**". Ebenso liegt die wahre (unbekannte) Regressionsgerade der Grundgesamtheit mit 100-α% im Konfidenzintervall. Für Prognosen ist der zu erwartende Fehler der **Einzelbeobachtung** wichtig. 100-α% der Einzelwerte werden im Konfidenzintervall der Einzelwerte erwartet. Wie man sieht, erweitert sich das Konfidenzintervall außerhalb des Messbereichs dramatisch, so dass sich allzu kühne Prognosen z.B. in die Zukunft verbieten.

Die Abbildung rechts zeigt die Regressionsgerade im X-Y-Koordinatensystem. Sie geht durch den Punkt **a** auf der Y-Achse und durch den Punkt $(\bar{x}, \bar{y})$. Die Messwerte y_i sind durch kleine Kreise, die Residuen e_i durch Striche dargestellt. Das Konfidenzintervall der wahren Geraden (wG) ist gestrichelt, das der Einzelwerte (Ew) ist gepunktet dargestellt.	

Die multiple Regression rechnet mit mehreren Einflussgrößen verschiedene lineare oder nichtlineare (quasilineare) Modelle.

Das lineare multiple Regressionsmodell $y = b_0 + b_1 x_1 + b_2 x_2 + ... + b_q x_q + e$	Beispiel nichtlineares Regressionsmodell $y = b_0 + b_1 x_1 + b_2 x_1^2 + b_3 x_1 x_2 + ... + b_5 x_2^2 + e$
	

Es gibt drei Hauptaufgaben der multiplen Regression:

1. **Prognose** (Vorhersage) von Y-Werten außerhalb des durch die x-Werte vorgegebenen Bereichs und/oder für neue Datenpunkte innerhalb des vorgegebenen X-Bereichs. Beispiele: Parameterpreisbildung, Schätzung der Energiekosten im nächsten Jahr auf der Basis der Produktionszahlen und Verbräuche in den vergangenen Jahren.
2. **Reproduktion** von Y-Werten exakt auf den Punkten des beobachteten X-Bereichs (Stützstellen). Es handelt sich hier um eine reine Datenreduktion (Regressionsparameter statt einzelner Y-Werte). Beispiel: Speicherung der Sicherheitspunkte der t-Verteilung für die Freiheitsgrade FG>5 mit einem Regressionsmodell der Form $t = b_0 + b_1*(1/FG) + b_2*(1/FG^2)$
3. **Finden von signifikanten Einflussgrößen** X: Beispiel: Welches sind die Haupteinflussgrößen auf den Ertrag einer neu entwickelten Rapssorte (Temperatur? Regenmenge? Kalk? Stickstoff?, ...)

Entsprechend den Hauptaufgaben sind verschiedene Regressionsalgorithmen zu empfehlen:

- Für Prognose (Vorhersage) wird das "Schrittweise Aufbauverfahren" oder das "Schrittweise Abbauverfahren" empfohlen
- Für die exakte Reproduktion der Y-Werte an den Stützstellen wird die "Regression mit allen Einflussgrößen" empfohlen.
- Für das Auffinden signifikanter Einflussgrößen wird das "Schrittweise Aufbauverfahren" oder das "Schrittweise Abbauverfahren" empfohlen

"Schrittweises Aufbauverfahren" bzw. "Abbauverfahren": Ein Signifikanztest (t-Test, F-Test) entscheidet über Aufnahme oder Verbleib einer Einflussgröße im Modell. Vorteile sind:

- Nur Einflussgrößen mit einem statistisch gesicherten Einfluss auf die Reduktion der Reststreuung werden in das Modell aufgenommen
- Eine Gruppe hoch korrelierter Merkmale wird durch ein Merkmal vertreten
- Es entsteht eine robuste Lösung, die auch bei moderaten Veränderungen in der Datenbasis noch Bestand hat

"Regression mit allen Einflussgrößen" ist ein Verfahren, bei dem nur Merkmale aus dem Modell entfernt werden, wenn eine so starke lineare Abhängigkeit der Merkmale diagnostiziert wird, dass numerische Instabilitäten auftreten. Der Vorteil ist: Für die Stützstellen (und nur für diese) lässt sich die Reststreuung maximal minimieren. Es hängt sehr vom Modell ab, ob die Zielgrößenschätzung auch für Werte außerhalb der Stützstellen noch vernünftige Zahlen liefert. Am besten testet man dieses aus, indem man selbst einmal die X-Werte leicht variiert und in das berechnete Modell einsetzt.

Erklärung der von der multiplen Regression benutzten und berechneten Größen:

Y	Das Zielgrößenmerkmal
X j	Einflussgrößenmerkmale (j = 1, 2, 3,...p) mit p= Einflussgrößenzahl
n	Auswertbare Punktzahl (Datensätze ohne Ausfall)
B	Multiples Bestimmtheitsmaß (multiples R^2), ein Maß für die Verbesserung der Vorhersage durch Kenntnis von X_1, X_2, ..., X_p. Es ist $0 \le B \le 1$. $B= SAQ_{Reg} / SAQ_{Rest}$. Dabei ist SAQ_{Reg} die Summe der Abweichungsquadrate aus Erwartungswerten und Mittelwert ($\sum(\hat{y}_i - \bar{y})^2$) und SAQ_{Rest} ist die Σe^2.
F	F-Testwert für R^2 bzw. B Die Nullhypothese ist Ho: B=0 (Kein modellmäßiger Zusammenhang zwischen Y und den Xj nachweisbar) mit F=B(n–k)/(1-B) und mit FG_1=1 und FG_2=n–k, k= Koeffizientenzahl einschließlich des b_o.
FG	FG=N-k, Freiheitsgrad der Reststreuung, k= Koeffizientenzahl einschließlich des b_o.
P-Wert	(oder KIW) Die Irrtumswahrscheinlichkeit bei einseitigem Test für die Ablehnung der Nullhypothese Ho:B=0 (H_A:B>0)
b_j	Koeffizient Der Zahlenwert des Regressions-Koeffizienten
s_{bj}	Stdabw. Die geschätzte Standardabweichung des Koeffizienten,
t_j	T-Wert t-verteilte Prüfgröße zum Test der Nullhypothese Ho: bj =0 (Koeffizient bj in der Grundgesamtheit Null?)
P-Wert	KIW Kritische Irrtumswahrscheinlichkeit bei zweiseitigem Test fuer die Ablehnung der Nullhypothese Ho:bj=0 (HA:bj<>0)
S_R	Reststreuung oder mittleres Residuum (mittlerer Fehler **e**)

Ausreißertest in den Residuen: Ausreißer sind einzelne Datenwerte, die sich außerhalb des üblichen Streubereichs bewegen. Sie machen jede Statistik kaputt. Z.B. ein Milliardär in Weigheim würde den Durchschnittsverdienst der Dörfler ins Astronomische heben. Zurück ins Blatt Autos.xls. b→Statistik →Grundlagen →Ausreißer →Variable *Residuen*, OK
Wir erfahren, dass Zeile 12 ein Ausreißer ist. Wir können die Zeile ausblenden und die Regression ohne diese Zeile wiederholen.

4. Ein paar Worte zu MS-WORD ®

Eigentlich ein überflüssiges Kapitel, da die meisten Studentinnen und Studenten WORD seit langem nutzen. MS-WORD ist ein Textverarbeitungsprogramm (Wysiwyg = What You see is what You get).

Wir starten MS-WORD und probieren einige Möglichkeiten aus:

---> Datei ---> **Seite einrichten**: Hoch- oder Querformat, Breite der Ränder ändern usw

Schriftarten: Times New Roman mit Serifen
Arial ohne Serifen
`Courier New mit Serifen, konstante Zeichenbreite`

Schriftgrößen: 8 Augenpulver

11 Normal für Zeitschriften
12 Normal für Thesisarbeiten

Schalter für: **fett** *kursiv* <u>unterstrichen</u>

Bündigkeit:

Linksbündig bei vielen Texten üblich, auch in Arbeiten, Briefen	**Rechtsbündig** eigentlich nur bei Abrechnungen, d.h. Zahlenkolonnen	**Zentriert** bei Überschriften, Gedichten, Abbildungen, Legenden	**Blocksatz** bei Spalten, auch in Büchern oder Thesisarbeiten. Wichtig ist die Silbentrennung.

Aufzählungen:

1. Gibt es in der nummerierten Form. Jedes ENTER 2. ruft einen neuen Punkt hervor mit 3. der automatischen Nummerierung 1, 2, 3, ...	• gibt es aber auch in der unnummerierten Form. • Auch hier erzeugt jedes ENTER einen • neuen Punkt

Die Rechtschreibehilfe (das ABC-Ikon) zeigt nacheinander alle Wörter im Absatz an, die nicht im Thesaurus stehen. Wenn man sich sicher ist, dass ein Wort richtig geschrieben wurde, kann man es dem Thesaurus hinzufügen.

Ausschneiden, Kopieren: Man färbt ein Textstück mit der linken Maustaste und geht dann auf die Schere oder rechts daneben auf das Kopiersymbol. Einen ausgeschnittenen oder kopierten Text kann man beliebig oft an fast beliebiger Stelle wieder **einfügen.** Man kann auch einen längeren Text kopieren, das Dokument verlassen und den Text dann in ein anderes Dokument einfügen.

Markieren langer Textabschnitte: Mit der Maus kann man Texte nur auf dem augenblicklichen Bildschirminhalt markieren. Berührt man den unteren oder oberen Rand, dann "geht die Post ab". Lassen Sie deshalb die Taste Großschreibung (⇑) gedrückt und navigieren Sie mit den 4 Kursortasten (Pfeiltasten) durch den Text, den Sie markieren wollen.

Rückgängig machen (der kleine gebogene blaue Pfeil nach links): Hat man Mist gebaut, kann man die letzten Aktionen rückgängig machen.

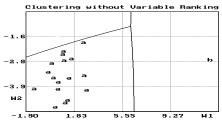
Tabelle einfügen: Tabellensymbol anklicken oder ---> *Tabelle*. Die Zeilen- und Spaltenzahl der Tabelle lassen sich leicht in der Menübox einstellen. Die Standardtabelle hat schwarze Linien für Zeilen- und Spaltenabgrenzung. Die Spaltenbreite kann man mit der Maus noch ändern. Die Zeilenhöhe passt sich automatisch dem Inhalt an. Man kann Zellen verbinden oder einzelne Zellen weiter teilen:

<table>
<tr><td colspan="2">ABBA</td><td></td><td></td></tr>
<tr><td colspan="2"></td><td colspan="2" rowspan="3">Hier wurden 6 Zellen verbunden: Die Zellen markieren --> Tabelle ---> Zellen verbinden</td></tr>
<tr><td colspan="2"></td></tr>
<tr><td>Diese Zelle</td><td>wurde geteilt</td></tr>
</table>

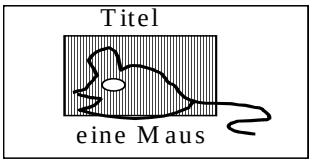
Mehrspaltiger Text: Spaltentextsymbol anklicken oder ---> *Format* ---> *Spalten* . Schreiben Sie aber den Text erst normal und bringen Sie ihn erst zum Schluss in die Spaltenform. Bei Formeln, Bildern und Tabellen muss man auf die Breite achten.

Formeleditor: Findet man unter →Einfügen →Objekt →Formeleditor oder unter dem Symbol $\sqrt{\alpha}$, z.B. als Übung $\Psi(x) = \frac{\sqrt{1-e^{-x}}}{\cos^2(x)} \left[\int_1^5 f(x)\, dx \right]$

Einfügen Seitenzahlen oben / unten usw
Datum / Uhrzeit, z.B. 15. Februar 2011
Sonderzeichen aus einer Tabelle, z.B. Φ oder ♠
Eine Graphik aus einer Datei, z.B.



Eine Datei, z.B. ein Messprotokoll oder ein anderes Dokument
`DASY-Protokoll vom 27.01.2007`
Ein Objekt, z.B. *Microsoft Word Graphik bzw. Microsoft Word Bild*. In dieser Graphik können Sie
farbige, beschriftete Zeichnungen aus Elementen erzeugen, die sich gut in den Text einpassen, auch z.B. in Tabellenfelder



Format **Zeichen** Verschiedene Unter- und ~~Durchstreichungen~~
Effekte z.B. mit Schatten
Design, z.B. **KAPITÄLCHEN** oder Mond
Farbe oder Animation, z.B. Funkeln oder Blinken
Skalierung, z.B. den Mond als Mond oder **Mond**

Absatz eingerückt (die Strecke ist einstellbar)
Zeilenabstand (einfach, $1^1/_2$, doppelt)
weitere Aufzählungen statt nummeriert oder gepunktet, z.B.
- ✓ Dies ist ein Thema
- ✓ und das ist noch eines

Rahmen, d.h. Texte wie Zeitungsannoncen, z.B.

Suche lieben Hund mit großen Ohren und braunen Augen, der mein Haus bewacht und der sich auch streicheln lässt

Extras z.B. ---> Sprache ---> Silbentrennung

IV Praktikum Prozedurale Programmierung

Teil 1: Win32-Konsolenanwendung

Sie arbeiten in 2-er-Gruppen. Sie holen sich von Frau Link im Dekanat (B-Bau, 2. Etage) Ihren Account, z.B. "ABC123". Das Praktikum findet im Raum A3.14 statt. Starten Sie einen freien Rechner. Geben Sie als Namen die 3 Buchstaben ein, z.B. ABC, als Passwort Ihren Account. Ihnen steht für das gesamte Studium an der FH auf Laufwerk "X:" viel persönlicher Speicherplatz zur Verfügung, der auch automatisch gesichert wird.

Sie richten zuerst einen neuen Ordner "IV" auf Laufwerk X: ein :
-->Start --> Arbeitsplatz --> Ordner (obere Leiste) --> X: --> Datei --> Neu --> "Neuer Ordner", diesen in "IV" umbenennen --> Fenster schließen

Programmieren und starten eines C-Programms:
--> Start --> Alle Programme --> MSC++6.0 --> MSC++6.0 --> Datei --> Neu --> Win32-Konsolenanwendung / Projekt z.B. "HalloWelt" / Pfad in "X:\IV" umändern / nochmals Win32-Konsolenanwendung anklicken -->"eine einfache Anwendung" -->Fertigstellen --> OK --> Fertigstellen (Es erscheint links oben "HalloWelt Klassen") --> Datei --> Öffnen --> HalloWelt öffnen --> HalloWelt.cpp öffnen. Es erscheint ein kleines Rumpfprogramm der Form

```
//HalloWelt.cpp .....
#include "stdafx.h"
int main(int argc, char* argr[])
{
        return 0;
}
```

Tragen Sie unter #include "stdafx.h" die Zeile ein: #include <Iostream.h>
Vor die return-Zeile setzen Sie: cout<<"Hallo, meine Welt, ich gruesse dich !!!";
--> Erstellen (obere Leiste) --> HalloWelt.exe ausführen, mit Ja antworten --> es erscheint das schwarze Ausgabefenster (Konsolenfenster) mit dem Text --> ENTER --> Datei --> **Arbeitsbereich schließen**, mit "Ja" antworten.

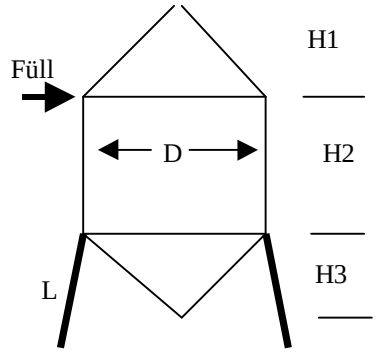
Wichtig: Immer bei Sitzungsende **Arbeitsbereich schließen,** bei Sitzungsbeginn **Arbeitsbereich öffnen.** Nie!!!!!! eine einzelne Datei öffnen.

Ein etwas anspruchsvolleres C-Programm , das die Eingabe eines Radiuswertes verlangt und das Kugelvolumen berechnet, hat dann etwa folgende Gestalt:

```
#include "stdafx.h"
#include <Stdio.h>
#include <Iostream.h>
#define PI 3.14159265
int main(.....
{       double R,Vol;                       // Reserviert Speicherplätze
        cout<<"Bitte Radius R eintippen : "; // Eingabehinweis für den Anwender
        cin>>R;                             // Erwartet Zahl und ENTER
        Vol=(4.0/3)*PI*R*R*R;               // berechnet Kugelvolumen
        printf("\n\nVolumen=%f",Vol);       // Gibt Text und Wert aus
```

```
        return 0;
}
```

Übung 1: Schreiben Sie ein C-Programm, das die Gesamtmasse des rechts abgebildeten Silos berechnet. Die 4 Stützen wiegen 78 Kg/m. Wandstärke ist 8 mm Stahl mit Dichte ρ=7800 Kg/m^3. Das Silo ist bis zum Pfeil gefüllt mit Zement der Dichte ρ=1900 Kg/m^3. Die Wandfläche eines Zylinders ist $A_z=\pi DH$, das Volumen $V_z=\pi D^2H/4$. Die Fläche eines Kegelmantels ist $A_k=\pi RS$ mit R=D/2 und $S=\sqrt{R^2+H^2}$, das Volumen $V_k=\pi D^2H/12$. Seien Sie bei den Berechnungen nicht zu pingelig, indem sie etwa beim Volumen auch noch die Wandstärke berücksichtigen wollen.



Übung 2: Füllen Sie in einer do-while-Schleife einen Vektor x[20] mittels Tastatureingabe von Zahlen mit minimal 3 und maximal 20 positiven Zahlen $x_1, x_2, ..., x_n$. Die Zahl n der eingetippten Zahlen ist im Programm durch Mitzählen zu bestimmen. Beenden Sie die Dateneingabe mit der Eingabe einer Null (0), die aber nicht zu den Daten selbst zählt. Berechnen Sie mittels einer for-Schleife das Produkt P und die Summe S der natürlichen Logarithmen der eingetippten Zahlen.

$$P=\prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdots x_n \qquad S=\sum_{i=1}^{n} \ln(x_i) = \ln(x_1)+\ln(x_2)+...+\ln(x_n)$$

Übung 3: Schreiben Sie ein Funktionspaar *dritte_wurzel* und *main* , das zu den Gewichten G von Stahlkugeln, welche Sie eintippen, Durchmesser D und größte Querschnittsfläche A der jeweiligen Kugel liefert (Stahldichte ρ=7800 Kg/m^3). Sie benötigen zur Berechnung von Radius R die dritte Wurzel w(x). Diese berechnen Sie in der Funktion nach Vorlage aus der Vorlesung iterativ nach der Formel $w_{neu}=(w_{alt} + x/w_{alt}^2)/2$ mit Startwert w_{alt}=1. Die Eingabe von immer neuen Gewichten G wird mit der Eingabe einer Zahl $G \geq 10^{25}$ beendet. Bei Kugeln gilt $G=\rho V$ mit $V=4\pi R^3/3$ und $A=\pi R^2$. Zur Berechnung von R lösen Sie Formel V nach R auf.

Teil 2: MFC Anwendungsassistent(.exe)

Übung 4: Greetings - mein erstes MFC-Programm unter Visual C++ 6.0 Windows

a) Start->alleProgramme->Visual C++ 6.0 ->Datei->Neu->1.Projektname: **Greetings**, 2.Pfad: X:\IV, 3. MFC Anwendungsassistent(.exe) ->OK->"SDI" anklicken, weiter->4*weiter->fertigstellen->OK

b) Unten links den Button "Kla.." anklicken, das "+" vor Greetingsklassen anklicken, Doppelklick auf CGreetingsDoc, gehe nach Zeile 14, Sp.2, dort folgenden Text eintippen (Soll uns die Adresse eines Grußtextes aus dem "Dokument" liefern):

```
protected: char  *m_Message;                    // definiert einen Textzeiger
public:    char *GetMessage()                   // definiert "Bringe-Funktion" als
           {return m_Message;}                  // Inline-Funktion
```

c) Datei schließen, speichern Ja, Unten links Button "Dat..", Klick auf das "+" vor Greetings Dateien, Doppelklick auf Quellcodedateien, Doppelklick auf GreetingsDoc.cpp, Zeile 33, dort den folgenden oder ähnlichen Grußtext im "Dolument" einfügen:

m_Message="Greetings, \n my friends!";

d) Datei schließen, speichern Ja, Doppelklick auf GreetingsView.cpp. Dieses Programm gibt alles auf unser weißes Fenster aus, Zeile 6 einfügen:

#include <Math.h>
#include <Stdio.h>

In etwa Zeile 63 nach der geschweiften Klammer dann folgenden Text einfügen:

```
//Hier beginnen BT/UV zu programmieren
// Zuerst den Begrüßungstext in 1. Zeile ausgeben, etwa die Mitte
pDC->TextOut(10,250, pDoc->GetMessage());
// Textkonstante ausgeben
pDC->TextOut(30,30,"Kugeltext");

// Definition einer Zeichenkettenvariablen
CString freier_text;
// Zuweisung eines Textes an die Variable
freier_text="Der Hund wedelt mit dem Schwanz";
//Ausgabe des Variableninhaltes (Text)
pDC->TextOut(30,100,freier_text);

//Zeichenketten "addieren" (geht nur mit CString) und das Ergebnis ausgeben
freier_text=freier_text+"haar";
pDC->TextOut(30,150,freier_text);

//Herstellen einer "Druckzeile" mit Zahlen und Ausgabe
char zeile[100];                                    // Speicher für eine Druckzeile
double a,b;                                         // 2 lokale Variable
a=sin(2.0);     b=cos(2.0);                         // irgendwelche Werte zuweisen
sprintf(zeile,"sin(2)=%6.3f  cos(2)=%6.3f",a,b);    // Formatierte Ausgabe nach zeile
pDC->TextOut(30,170,zeile);                         // Ausgabe der zeile

//Das Malen einer schwarzen Linie
pDC->MoveTo(50,50);        pDC->LineTo(100,100);

//Das Malen einer roten Linie aus einzelnen Pixeln (RGB(rot,grün,blau))
int j;
for(j=10;j<100;j++) { pDC->SetPixelV(j+200,200,RGB(255,0,0)); }

// Zeichne Kreisbogen (gegeben durch Begrenzungsrechteck, Punkt oben links x,y
//          Punkt unten rechts des Begrenzungsrechtecks  x,y
//           Startpunkt Bogen x,y,           Endpunkt Bogen   x,y
pDC->Arc(30,40,  100,100,   35,45,   70,70);   //Zeichne den Bogen
```

e) -> Erstellen -> Kompilieren -> Erstellen -> Ausführen -> Beenden oder X-Button

f) „***Spielen***“ Sie mit dem Programm: Verändern Sie die Farbwerte im RGB-Tripel, die Zielposition der Texte und anderes mehr, damit Sie die Wirkung der einzelnen Anweisungen und Funktionen erkennen. Anweisungen und Funktionen der gezeigten Art können in der Klausur verlangt werden.

Übung 5: Kosinus-E-Kurven-Tabelle im Viewfenster erzeugen (Projektname **Tabelle**)
Diese Übung ist fakultativ, d.h. nur für Interessierte gedacht und **nicht klausurrelevant**. Da das gezeigte Programm sehr viele Details enthält, die in „richtigen Anwendungsprogrammen“ auftreten, ist es als Vorlage für Studierende gedacht, die im Rahmen einer Studienarbeit oder Bachelorthesis oder im Beruf sich in C++ vertiefen müssen.

Wir wollen ein MFC-Programm schreiben, das im Endausbau folgendes leisten soll:

- Eine Tabelle mit Titel, Spaltenbezeichnungen und Werten zeigt. Die Spalten sind x, cos(x), y(x) mit der Funktion $y(x) = e^{-ax} - e^{-by}$
- Ein Fenster mit Scrollbalken benutzt
- Von einer Datei mit Extension *.dat die beiden Koeffizienten a, b der Kurve y(x) liest
- Mit einer Dialogbox Schrittweite Dx und Endwert Xe des Tabellen-x vorgeben
- Wir können die Tabellenwerte auf eine Datei ausgeben, z.B. X:\Tabelle.dat

a) Scrollbalken am Viewfenster: ->TabelleView.h-> aus *CView* wird ***CScrollView*** ->TabelleView.cpp-> alle 8 Auftreten von *CView* ändern in ***CScrollView*** ("alle ersetzen") ->TabelleView.h->Ansicht->Klassen-Assistent-> Nachrichtenfenster->*OnInitialUpdate* ->hinzufügen-> *Code bearbeiten*->Nach der Zeile *//ToDo* einfügen:

SIZE Size={640, 2500}; // Gibt Breite und Höhe des neuen Fensters an
SetScrollSizes(MM_TEXT, Size); // Übergibt Maßstab und Fenstermaße

->Speichern->Ausführen: Das Viewfenster hat jetzt Scrollbalken

b) Dialogbox zur Eingabe von Schrittweite Dx und Endwert Xe einrichten: ->TabelleDoc.cpp ->Einfügen->Ressourcen->Dialog->Neu->aus der Toolbar 2 Felder "ab" und 4 Felder "Aa" in die Dialogbox ziehen. In jedem Feld: ->linker Mausklick->rechterMausklick->Eigenschaften: Bei Texten statt "static" Ihren Text, IDC_EDIT1 umbenennen in **IDC_DX**, IDC_EDIT2 in **IDC_XE**.->Die Dialogbox ist fertig->rechter Mausklick auf graue Fläche->Klassenassistent->Klasse hinzufügen->Neu->als Namen *CDxXe* angeben->Fortfahren->Membervariablen der neuen Klasse CDxXe definieren->im großen Fenster Zeile IDC_DX anklicken->neue Variable->als Namen ***m_Dx*** nehmen, ähnlich für Xe ->Ansicht->Klassenassistent->Klasse CDxXe -> im mittleren Fenster WM_INITDIALOG anklicken->Funktion hinzufügen (Liefert OnInitDialog)->WM_PAINT anklicken->Funktion->liefert OnPaint->damit ist die Dialogklasse CDxXe perfekt.

c) Klasse TabelleDoc.h erweitern

```
class CTabelleDoc : public CDocument
{ .........
// Attribute
public: CString m_Dx,m_Xe; //Schrittweie, Endwert als Zeichen
        double dx, xe, a, b; //Zahlen
```

d) Programm TabelleDoc.cpp Den alten Inhalt völlig löschen, den neuen Inhalt vollständig von meinem USB-Stick importieren (Einfügen von Datei) und nur die mit

```
// <<<<<<=======   markierte Zeile ändern (anderer Pfadname!)
#include "DxXe.h" //Dialogklasse bekannt machen
#include "Math.h" // cos, exp bekannt machen
#include "Stdio.h" //FILE, fopen, fprintf bekannt machen

CTabelleDoc::CTabelleDoc()
```

```
{    // ZU ERLEDIGEN: Hier Code für One-Time-Konstruktion
einfügen
     m_Dx="0.1";    //Defaultwerte setzen
     m_Xe="1.0";    //Defaultwerte setzen
     a=0.07;   b=0.235;  //Defaultwerte setzen
}
.............
BOOL CTabelleDoc::OnNewDocument()
{ .............
     // ZU ERLEDIGEN: Hier Code zur Reinitialisierung einfügen
     // (SDI-Dokumente verwenden dieses Dokument)
     FILE *ausgabe; // Dateihandle für Tabellenausgabe
     char *pEnd;    // Fehlermelder beim Zahlen umwandeln
     double x,cosi,y;    // Für das Berechnen der Tabelle
     //Zuerst dx und xe mit der Dialogbox lesen
     CDxXe DxXeDlg;  //Deklaration einer Dialogbox
     //Initialisiere die Box mit den Defaultwerten
     DxXeDlg.m_Dx=m_Dx;
     DxXeDlg.m_Xe=m_Xe;
     //Rufe Dialog-Box auf
     if(DxXeDlg.DoModal()==IDOK)   //Warte bis OK kommt
     {    m_Dx=DxXeDlg.m_Dx;       //Übernimm die Daten
          m_Xe=DxXeDlg.m_Xe;       //aus der Dialogbox
          dx=strtod(m_Dx,&pEnd);   //Kette in Zahl wandeln
          xe=strtod(m_Xe,&pEnd);   //Kette in Zahl wandeln
          // max. 101 Zeilen soll die Tabelle lang sein
          if( (100.0*dx)<xe ) xe=100.0*dx;
     }    // Ende des Dialogboxaufrufs und -auswertung
     // Eingabe der a,b- Daten für y(x) von einer Datei
     // Diese muss zwei Zeilen mit je einer Zahl enthalten
     // Definition eines Filedialogs für Datei öffnen
     // "true" steht für "Öffnen"
     CFileDialog fileDialog(true,NULL,NULL,NULL,
     "Datendateien (*.dat)|*.dat|Alle Dateien (*.*)|*.*||");

     // Filedialog starten und auf OK warten (Signal IDOK)
     if(IDOK==fileDialog.DoModal())
     {    DeleteContents();   //alte Daten löschen
     try //Mache einen Leseversuch mit der Datei
     {    //Definiere ein File-Objekt vom Typ Textdatei
          CStdioFile file(fileDialog.GetPathName(),
                                  CFile::typeText);
          CString line;  // Speicher für eine Zeile der Datei
          int lileng, anzkoef;  //Zeilenlänge, Koeffiz.-Zähler

          // ReadString liest bis einschließlich "End-of Line",
          // liefert eine Zeichenkette ohne "EOL-Zeichen" ab.
          // Datenzeilen lesen bis zwei Koeffizienten oder
          // Dateiende gefunden wurde. Leerzeilen überlesen
          // Fehlerhafte Zahlen werden als Zahl 1 interpretiert
          anzkoef=0; // zählt die gelesenen Koeffizienten a,b
          while((file.ReadString(line))&&(anzkoef<2))
```

```
          { //Zahl als Kette aus der Zeile ausschneiden
            char zkette[50];  //Speicher für eine Kette
            double koef;      //Speicher für ein Zahl
            int i,j;          //Zwei Zeichenzähler
            lileng=line.GetLength(); //Ermittle Zeilenlänge
            if(lileng>0) // Wenn keine Leerzeile, dann ...
           { j=0; i=0;
            //Übergehe Blanks und Tabs bis zur ersten Ziffer
            while((line[j]==' ')||(line[j]=='\t')){j++;}
            //Nimm alle Zeichen ungleich Blank oder Tab
            while((line[j]!=' ')&&(line[j]!='\t'))
               {zkette[i++]=line[j++];}
            zkette[i]='\0';   //Abschlussnull als Kettenende

            //pEnd zeigt auf das Ende des konvertierten
            //Teils der Zeichenkette. Bei einer gültigen Zahl
            //sollte das auch das Ende der Zeichenkette sein
            koef=strtod(zkette,&pEnd);  //Wandle Kette in Zahl
            if( (pEnd-zkette)!=i) {koef=1;} //im Fehlerfall 1
            //Abspeichern des Koeffizienten als a bzw. b
            if (anzkoef==0) a=koef;
            if (anzkoef==1) b=koef;
            anzkoef++;  //gelesene Koeffizienten zählen
            } //Ende des Blockes if(lileng>0...
          } // Ende des Blockes while....
          file.Close();
      }//End try-Block. Es folgt der CatchTeil
       catch(CFileException *e)
       {e->ReportError();
        e->Delete();
        return (FALSE);
      }//End catch
      } //End if(IDOK.....
      // Ausgabe der Tabelle auf z.B. Datei A:Tabelle.dat
      //Ausgabedatei öffnen für Schreiben
      ausgabe=fopen("X:\\tabelle.dat","w");  <<<<<<<==========
      x=0; // x-Startwert für Schleife
      do
      {    cosi=cos(x);
           y=exp(-a*x) - exp(-b*x);
           //formatierte Ausgabe in die Datei
           fprintf(ausgabe,"%10.2f%15.5f%15.5f\n",x,cosi,y);
           x+=dx;    // x erhöhen
      }while(x<(xe+dx/2));     //Schleifenende-Test
      fclose(ausgabe);
      UpdateAllViews(NULL);
      return TRUE;
}.......... der Rest bleibt unverändert
```

e) Programm TabelleView.cpp vervollständigen
Zuerst **`#include "Math.h"`** zu den include-Zeilen hinzufügen

```
void CTabelleView::OnDraw(CDC* pDC)
{    CTabelleDoc* pDoc = GetDocument();
     ASSERT_VALID(pDoc);
     // ZU ERLEDIGEN: Hier Code zum Zeichnen der
     urspr....hinzufügen
     RECT ClientRect; //Neues Objekt einer Datenstruktur
     GetClientRect(&ClientRect); //Fensterdaten ermitteln
     CString sdx,sxe;    //lokale Kettenvariable deklarieren
     sdx=pDoc->m_Dx;     //Hole Originalketten aus Dokument
     sxe=pDoc->m_Xe;
     pDC->TextOut(0,0,"Schrittweite="+sdx); //Ausgabe ...
     pDC->TextOut(0,20,"Endwert="+sxe);     //.. ins Fenster
     pDC->TextOut(200,0,"Tabelle für Frau ..."); //Titelzeile
     pDC->TextOut(150,30,"  x   cos(x)   y(x)"); //Kopfzeile

     char zeile[100];    //Speicher für eine Textzeile
     int i;              //Zeilenzähler
     double x,dx,xe,cosi,y,a,b;   //lokale Zahlenspeicher
     dx=pDoc->dx;   //Hole Originalwerte aus dem Dokument
     xe=pDoc->xe;             //Hole Originalwerte
     a=pDoc->a; b=pDoc->b;    //Hole Originalwerte
     i=0;       x=0;       // i- und x-Startwert für Schleife

     do             //Begin einer do-while-Schleife
     {    cosi=cos(x);
          y=exp(-a*x) - exp(-b*x);
          //formatierte Ausgabe auf eine Zeichenkette
          sprintf(zeile,"%10.2f%15.5f%15.5f",x,cosi,y);
          // Ausgabe der hergestellten "Zeile"
          pDC->TextOut(150,60+i*20,zeile);
          i++;      x+=dx;    //Zeilen zählen, x erhöhen
     }while(x<(xe+dx/2));     //Schleifenende-Test

......... der Rest bleibt unverändert
```

f) Stellen Sie mit dem Editor aus Zubehör eine Datei, z.B. AB.dat her (bzw. kopieren Sie sie von meiner Diskette) die in zwei Zeilen die zwei Zahlen a und b enthält, z.B.:

```
0.789
0.034
```

Bringen Sie die Datei in Ihren IV-Ordner und testen Sie dann das Programm Tabelle. Übernehmen Sie die Ausgabedaten in EXCEL und machen Sie dort eine Linien-XY-Grafik.

g) Machen Sie folgende Änderungen am Programm:

- Vergrößern Sie die Zahl der möglichen Tabellenzeilen auf 500.
- Geben Sie in TabelleView zusätzlich Mittelwert, Maximum und Standardabweichung σ_{n-1} der y-Werte aus.

Teil 3: Java in Entwicklungsumgebung BlueJ

Übung 6: Ticketautomat Java-Übung im System BlueJ

a) Installieren und testen Sie die Klasse

-> BluJ -> Project -> New Project -> Eigene Dateien -> ... auf Samba ... -> IV -> Dateiname: **Ticketautomat** -> Create -> Edit -> New Class -> Class Name: **Ticketautomat** -> O.K. -> r. Klick auf die Klasse -> Editor -> Eintippen der Klasse "Ticketautomat" aus der Vorlesung -> Compile -> Fehler ausmerzen -> Compile -> bis "no Syntax Errors" -> Close -> r.Klick auf die Klasse -> New Ticketautomat -> Preis eintippen -> O.K.
-> r.Klick auf den roten "Ticketautomat 1" -> geldEinzahlen() -> eine Zahl eintippen -> O.K.
.-> r.Klick auf den roten "Ticketautomat 1" -> ticketDrucken() -> O.K.

b) Verändern Sie die Klasse Ticketautomat so, dass der Automat
nur ein Ticket druckt, wenn genügend Geld eingeworfen wurde
Rausgeld ausgibt (zusätzliche Zeile auf dem Ticket)
bei Methode "Abbrechen" das eingezahlte Geld rausgibt (Druckzeile "Rausgeld")

Ubung 7: Hefewachstum Java-Übung mit dem System BlueJ

Lösen Sie das System der beiden gekoppelten DGL numerisch mit dem Euler-Verfahren, d.h. berechnen Sie die beiden Lösungsfunktionen y(t) und N(t) aus den Anfangswerten. Für die Graphik benutzen Sie die importierte Klasse "Leinwand" (von meiner Diskette):

$$y' = (b N / V - s) y \qquad N' = -(C + r b N / V) y$$

y [Kg]	Hefemenge im Fermenter.	Startwert y0 = 20 Kg
N [Kg]	Nährstoffmenge (Substrat, z.B. Zucker).	Startwert N0 = 10.000 Kg
V [l]	Fermentervolumen	V = 100.000 Liter
b [l/(h Kg)]	Theoretische Vermehrungsrate bei Nährstoffkonzentration 1 Kg/Liter	b = 0.2 pro Stunde und Kg/l
s [1/h]	Sterblichkeit der Hefezellen	s = 0.005 (0,5 % pro Stunde)
Cmax [Kg/(Kg h)]	Maximaler Nährstoffverbrauch in Kg pro Kilo Hefe und pro Stunde	Cmax = 0.025
C [Kg/(Kg h)]	tatsächlicher Nährstoffverbrauch. Bei Überangebot an Zucker (N/y > Cmax) ist C=Cmax, Bei Unterangebot (N/y ≤ Cmax) ist C = N/ y .	
r [Kg/Kg]	Nährstoffverbrauch [Kg] für die Zunahme der Hefemenge um 1 Kg	r = 2.5 Kilo Zucker pro Kilo Hefe
t [h]	Zeit mit Einheit auf der t-Achse	
dt [h]	Zeitschritt der Euler-Integration	dt = 1/ 3600 Stunde (1 Sekunde)
te [h]	Ende der Fermentation	te = 500 Stunden

massstaby=0.2, massstabN=0.03 bei einer Graphik mit 400 Pixeln Höhe

Zeichnen Sie das Achsenkreuz, beschriften Sie die t-Achse mit der gewandelten Zahl te.
Zeichnen Sie die Kurve y(t) schwarz, N(t) rot, Überschrift "Hefewachstum blau.

Praktikumsanleitung "Vom Experiment zum Dokument"

Praktikum Laborbesuch im AT-Labor Raum A-Bau, 1. UG

Sie bilden 2 oder 3 Gruppen, die jeweils 45 bzw. 30 Minuten im AT-Labor im A-Bau, 1. Untergeschoss, verbringen.

- Es wird eine Aufheiz-Abkühlmessung mit dem PREMA-Messcomputer gestartet. Am Ende der Vorführung werden die Daten gespeichert.
- Es wird eine Messung am Datenlogger gestartet mit Wind-, Temperatur- und Stromdaten. Jeder darf den Fön bedienen. Die Daten werden gespeichert.
- Es wird LabView gestartet und eine Temperaturkurve bei abwechselnd ein- und ausgeschalteter Heizung registriert. Die Daten werden gespeichert.

Praktikum Datenbank mit ACCESS und mit EXCEL

(MA-PC-Hall A3.14) Sie bilden Praktikumsgruppen à 2 Personen für die gesamte Zeit des IV-Praktikums. Dafür geht eine Liste um. Nur eine einzige Dreier- bzw. eine Einergruppe ist gestattet, wenn die Zahl der Personen ungerade ist.
Sie legen nach der Anleitung aus der Vorlesung in Access eine Datenbank *Gruen* an, aber mit etwas mehr Blattdaten als in der Vorlesung (pro Ernte etwa 10 Blätter).
Ligusterblätter ohne Stiel sind etwa 30 - 60 mm lang und 10 - 25 mm breit.
Buchenblätter ohne Stiel sind etwa 60 - 90 mm lang und 40 - 70 mm breit.

Als Ergebnis Ihrer Access-Sitzung haben Sie einen **Excel-Bericht** und eine Datei **Blattdat.txt** als Ergebnis einer Abfrage.

Legen Sie jetzt nach dem Vorbild im Skript eine ähnliche Datenbank in Excel an und erzeugen Sie ebenfalls eine Auswahl. Kopieren Sie diese Auswahl in ihre Hausarbeit.

Alle restlichen Praktika: Berechnungen mit EXCEL

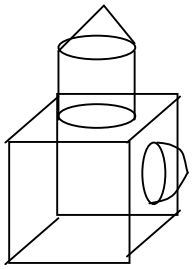
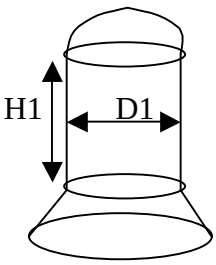
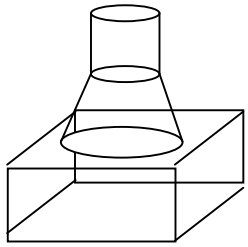
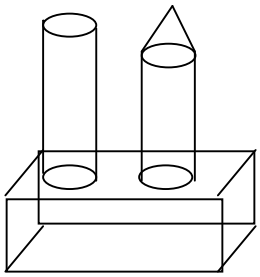
Die 5 weiteren Excelaufgaben, die Sie zu lösen haben, unterscheiden sich leicht von Gruppe zu Gruppe. Die Gruppennummer **GNr** legt z.B. die 1. Aufgabe fest. Bei den Aufgabe 2-5 wird der 1. oder 2. Buchstabe Ihres Vor- bzw. Nachnamens genommen (ä=a, ö=o, ü=u, sch=s, ß=s).

Besorgen Sie sich einen Account für die MuV-PC-Hall (Raum A3.14) im Dekanat.

Legen Sie auf Ihrer X-Platte (samba) einen Ordner ***IV*** an, und in diesem Ordner ein WORD-Dokument ***Hausarbeit Excel***. Die Überschrift lautet *Hausarbeit Excel.* Darunter schreiben Sie die Gruppennummer GNr, dann Ihre Namen und fügen ihre Passbilder ein (2 BearbeiterInnen pro Hausarbeit. Ausnahme siehe oben).

Der Lösungsweg der Aufgaben wird in der Vorlesung gezeigt. Sie bearbeiten ähnliche Aufgaben, d.h. Sie müssen den Lösungsweg anpassen. Jede Aufgabe beginnt im WORD-Dokument mit einer Überschrift

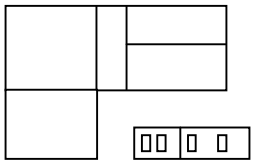
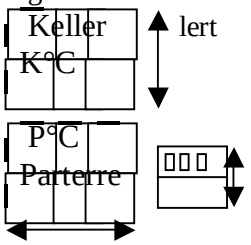
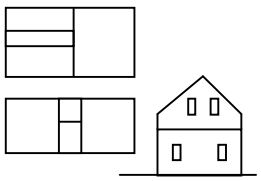
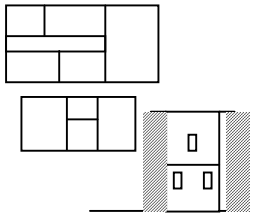
Aufgabe 1: Behälter. Sie entnehmen nach Gruppennummer GNr Ihre Vorlage

GNr 1, 5, 9, 13, 17, 21, 25, 29 Kegel, Zylinder, Kubus, Halbkugel	GNr 2, 6, 10, 14, 18, 22, 26, 30 Halbkugel, Zylinder, Kegel ohne Spitze	GNr 3, 7, 11, 15, 19, 23, 27, 31 Zylinder, Kegel ohne Spitze, Quader	GNr 4, 8, 12, 16, 20, 24, 28, 32 Kegel, Zylinder, Quader
			

a) Konstruieren Sie Ihren Behälter, indem Sie die verlangten geometrische Formen, wie Quader, Kugel, Halbkugel, Zylinder, Kegel usw. benutzen. Bemaßen Sie ihre Konstruktion mit Buchstaben, wie es im 2. Behälter angedeutet wird. Machen Sie mit *Microsoft Word-Bild* eine Abbildung mit dieser Bemaßung. Diese Abbildung muss in keiner Weise maßstabgetreu sein, sondern soll lediglich als Konstruktionsskizze verstanden werden. Fügen Sie das Bild in das rechte Feld einer 2-spaltigen Tabelle ein. In das linke Feld schreiben Sie etwas über den Bestimmungszweck Ihrer Konstruktion, z.B. „Zementsilo auf einer Baustelle“ oder „Abkühlbehälter für Kochwurst“ und erklären Sie die Maße, z.B. *H1=Höhe des Zylinders, D1=Durchmesser des Zylinders*.
b) Bauen Sie eine Exceltabelle mit maximal 7 Spalten Breite (diese Breite lässt sich gerade noch in das WORD-Dokument kopieren) und beliebig vielen Zeilen auf, die den Rauminhalt **V** und die Oberfläche **A** Ihrer Konstruktion berechnet. Benennen Sie alle Zahlen und weisen Sie ihnen sinnvolle Werte zu. Bringen Sie durch helle Hintergrundfärbung Struktur in die Tabelle, zum Beispiel Eingabefelder hellgrün, Kommentare weiß, Ergebnisse hellgelb.
c) Besorgen Sie sich die Dichten von Portlandzement, Weizen, Gips, und Kies. Schreiben Sie diese Dichten in Ihre Heimarbeit. Berechnen Sie mit dem Volumen **V** und einer der Dichten die Maximalmasse der Füllung. Besorgen Sie sich die Dichte ρ von Baustahl und berechnen Sie mit einer angenommenen Materialdicke **d** und der Oberfläche **A** die Leermasse ihrer Konstruktion (Leermasse = A d ρ). Berechnen Sie die maximale Gesamtmasse. Führen Sie die Berechnung aus und kopieren Sie die Tabelle in ihre Heimarbeit.
d) Machen Sie dieselben Berechnungen mit dem Taschenrechner und geben Sie diese Werte unter der Überschrift „Kontrollrechnung mit dem Taschenrechner“ unter der Exceltabelle in die Heimarbeit ein (Grund: Ein Ingenieur muss in der Praxis für seine Berechnungen gerade stehen, deshalb Kontrollrechnung bei eigenen Formeln üben.)

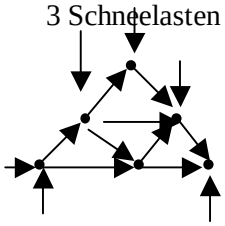
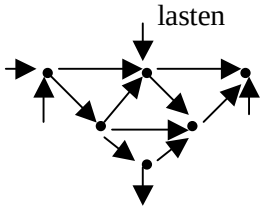
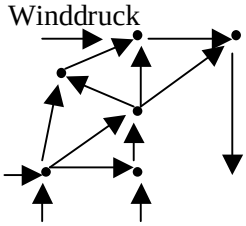
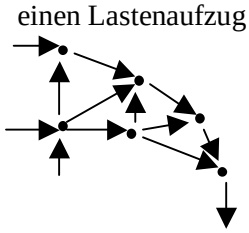
Aufgabe 2: Wärmeverbrauch. Sie entnehmen nach dem 1. Buchstaben des ersten Vornamens Ihre Vorlage. Zeichnen Sie mit Microsoft Word-Bild Ihr Haus mit dem (den)

Grundrissen der Geschosse und angedeuteten Fenstern, außerdem mindestens eine Seitenansicht. Übernehmen Sie das Bild in ihre Hausarbeit.

1. Buchstabe Vorname ist A-F	1. Buchstabe Vorname ist G-L	1. Buchstabe Vorname ist M-R	1. Buchstabe Vorname ist S-Z
Bungalow nicht unterkellert 	Bungalow unterkellert 	Einfamilienhaus 	Reihenmittelhaus 

a) Benennen Sie die Maße der Geschosse ihres Hauses mit Buchstaben. In jedem Raum eines Geschosses nehmen Sie dieselbe konstante Temperatur an, z.B. Temperatur P im Parterre. Legen Sie die Fensterzahl und die Fensterflächen fest. Ebenso für Türen.
b) Suchen Sie im Internet die Bodentemperatur (Sie ist gleich der mittleren Außentemperatur.) Suchen Sie ebenfalls dort die U-Werte (Wärmedurchgangswerte, früher k-Werte) für Ihr Mauerwerk und Ihre Fenster und Türen.
c) Berechnen Sie in einer farbig strukturierten Exceltabelle die Wärmeabgabeflächen jedes Geschosses getrennt nach Mauerwerk und Fenster- bzw. Türfläche. Wärmefluss in Watt ist $\dot{Q} = U \cdot A \cdot (\vartheta_i - \vartheta_a)$. Dabei ist A die Fläche mit Wärmedurchgangszahl U, θ_i die Innentemperatur, θ_a die mittlere Außentemperatur. Die Wärme fließt von der höheren zur niederen Temperatur. Berechnen Sie alle Wärmeflüsse nach außen durch die Wände, zum Erdboden, zum Dach und getrennt davon die durch die Fenster und Türen.
d) Berechnen Sie die Wärmemenge pro Jahr in KWh. Besorgen Sie sich im Internet den Heizwert (Energieinhalt) und den Preis von einem Liter Heizöl und berechnen Sie den Ölverbrauch und die jährlichen Ölkosten. Kopieren Sie die Tabelle in Ihre Hausarbeit und schreiben Sie einige Bemerkungen zur Möglichkeit von Energieeinsparung.
e) Schreiben Sie mit dem Formeleditor obige Wärmetransportgleichung und ihre Erläuterung in ihre Hausarbeit. Kopieren Sie aus dem Internet eine kurze Erläuterung des Unterschiedes zwischen Heizwert und Brennwert (max. ½ Seite).

3. Aufgabe: Stabwerk. Sie entnehmen nach dem 2. Buchstaben des ersten Vornamens Ihre Vorlage. Zeichnen Sie mit Microsoft Word-Bild Ihr Stabwerk größer und farbig mit Benennungen der Stäbe, Knoten, Auflagerkräfte und Belastungen. Übernehmen Sie das Bild in ihre Hausarbeit.

2. Buchstabe Vorname ist A-F	2. Buchstabe Vorname ist G-L	2. Buchstabe Vorname ist M-R	2. Buchstabe Vorname ist S-Z
Dachbinder mit 3 Schneelasten 	Brücke mit 2 Mittellasten 	Kran mit Last und Winddruck 	Wandhalterung für einen Lastenaufzug 

a) Stellen Sie nach dem Vorbild der Vorlesung eine Exceltabelle für die 6 Knotenkoordinaten und die 9 Winkel der 9 Stäbe mit ihren Sinus- und Kosinuswerten auf. Dann bestimmen Sie die $2 \times 6 = 12$ Knotengleichungen und programmieren die 12×12-Matrix der Koeffizienten und die rechte Seite als Vektor mit den Lasten. Berechnen Sie die Druck- und Zugkräfte in den Stäben und Auflagern als Lösung des Gleichungssystems.
b) Kopieren Sie nur die Knotenkoordinaten, die Stabwinkel und die berechneten 12 Kräfte in Ihre Hausarbeit. Schreiben Sie neben die Kräfte, ob es Druck- oder Zugkräfte sind.

4. Aufgabe: Hefewachstum. Sie entnehmen nach dem 1. Buchstaben des ersten Nachnamens Ihre Vorlage.

1. Buchstabe Nachname ist A-F	1. Buchstabe Nachname ist G-L	1. Buchstabe Nachname ist M-R	1. Buchstabe Nachname ist S-Z
Feed: Im Abstand von 2 Tagen wird jeweils 1/3 der vorgesehenen Substratmenge S_o eingesetzt. Wachstum μ: Sie verwenden die Mortalitätsformel $M(c_H) = (c_{HMax} - c_H)^{0,3} / c_{HMax}^{0,3}$	Feed: Jede Stunde wird die Menge $S_o/100$ eingesetzt, bis nach 100 Stunden die geplante Substrat-menge S_o erreicht ist. Wachstum μ: Sie verwenden die Mortalitätsformel $M(c_H) = (c_{HMax} - c_H)^{0,2} / c_{HMax}^{0,2}$	Feed: Die halbe Startmenge S_o wird sofort zugegeben, die zweite Hälfte wird auf 150 Stunden gleichmäßig verteilt. Wachstum μ: Sie verwenden die Mortalitätsformel $M(c_H) = (c_{HMax} - c_H) / c_{HMax}$	Feed: 1/10 der Start-menge S_o wird sofort zugegeben, der Rest wird auf 80 Stunden gleichmäßig verteilt. Wachstum μ: Sie verwenden die Mortalitätsformel $M(c_H) = (c_{HMax} - c_H)^{1/2} / c_{HMax}^{1/2}$

5. Aufgabe: Multiple Regressionsanalyse. Sie entnehmen nach dem 2. Buchstaben des ersten Nachnamens Ihre Vorlage.

1. Buchstabe Nachname ist A-F				1. Buchstabe Nachname ist G-L				1. Buchstabe Nachname ist M-R				1. Buchstabe Nachname ist S-Z			
Zielgröße sind Kosten Ko, Einflussgrößen sind Stromstärke Str, Spannung Spa, Gewicht Gew				Zielgröße ist Zugfest-igkeit Z, Einfluss-größen sind Press-druck Pd, Dauer t, Leimmenge L				Zielgröße ist Dichte D, Einflussgrößen sind Kiesanteil K, Zementanteil Z, Mischdauer t				Zielgröße ist Boden-wertzahl Bz, Einfluss-größen sind Bereg-nung Be, Humus Hu, Tonmenge Ton			
Ko	Str	Spa	Gew	Z	Pd	t	L	D	K	Z	t	Bz	Be	Hu	Ton
3,95	0,49	0,40	0,72	2,51	0,18	0,60	0,66	0,84	0,02	0,89	0,04	0,78	0,06	0,55	0,35
2,93	0,25	0,41	0,27	0,12	0,03	0,04	0,50	2,01	0,50	0,42	0,45	1,03	0,94	0,77	0,78
3,16	0,29	0,45	0,02	3,98	0,84	0,69	0,58	1,15	0,25	0,73	0,16	1,25	0,24	0,98	0,97
4,87	0,78	0,21	0,02	1,22	0,36	0,19	0,72	3,05	0,90	0,67	0,86	0,80	0,59	0,68	0,10
3,62	0,26	0,77	0,92	0,24	0,11	0,05	0,74	1,14	0,09	0,80	0,29	0,51	0,83	0,26	0,04
5,55,	0,81	0,45	0,04	0,61	0,07	0,16	0,67	2,01	0,24	0,99	0,93	0,64	0,60	0,43	0,04
4,89	0,74	0,30	0,71	4,89	0,96	0,87	0,65	2,10	0,79	0,70	0,04	0,93	0,87	0,88	0,02

6.66 0,99 0,69 0,18	1,11 0,56 0,08 0,50	1,72 0,57 0,77 0,02	0,70 0,88 0,44 0,27
3,82 0,55 0,18 0,40	3,68 0,70 0,66 0,41	2,09 0,53 0,64 0,48	1,09 0,60 0,93 0,52
3,91 0,51 0,34 0,76	2,85 0,65 0,49 0,86	1,18 0,15 0,12 0,18	0,55 0 45 0,07 0,75
5,49 0,81 0,47 0,82	1,49 0,06 0,39 0,53	2,33 0,66 0,32 0,53	0,49 0,50 0,19 0,15
5,71 0,72 0,80 0,70	3,38 0,81 0,53 0,23	2,20 0,77 0,84 0,19	1,05 0,73 0,69 0,98
2,01 0,08 0,32 0,79	2,37 0,39 0,44 0,11	1,37 0,18 0,89 0,33	0,52 0,12 0,09 0,53
6,33 0,90 0,71 0,66	4,48 0,60 0,90 0,24	1,94 0,31 0,75 0,77	0,89 0,30 0,60 0,61

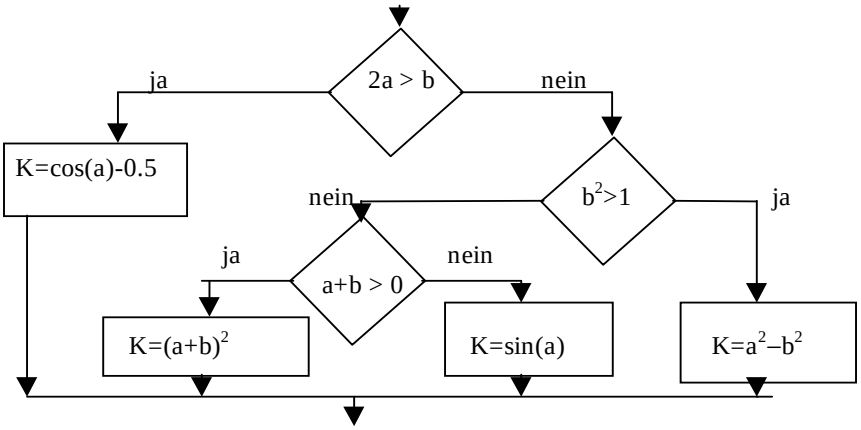
a) a) Schreiben Sie eine Textdatei mit dem Editor, die Ihre Daten enthält. Zwischen den Zahlen einer Zeile setzen Sie einen Tabstopp, am Ende der Zeile ENTER. Speichern Sie diese Datei unter dem Namen *Regressionsdaten Hausarbeit.txt* auf ihre X-Platte.
b) Importieren Sie die Regressionsdaten in eine Exceltabelle. Führen Sie nach dem Vorbild der Vorlesung bzw. des Skriptes eine Regressionsanalyse durch. Bestimmen Sie die t-Werte für die 4 Koeffizienten (Konstante, 3 Regressionskoeffizienten)
c) Wiederholen Sie die Regression unter Auslassung des nicht signifikanten Koeffizienten. Ist die Konstante b_o nicht signifikant, dann reicht eine Wiederholung der Regression, wobei das erste WAHR in ein FALSE umgewandelt wird (Modell ohne Konstante). Ist jedoch einer der 3 Regressionskoeffizienten nicht signifikant, dann müssen Sie die Datenreihen kopieren. Dabei lassen Sie die Einflussgröße weg, die nicht signifikant ist (reverse Reihenfolge der Koeffizienten beachten!!!). Die Datenreihen der verbliebenen zwei Einflussgrößen müssen direkt nebeneinander stehen.
d) Programmieren Sie die Ausgabe der Hypothesen Ho bzw. Ha.
e) Kopieren Sie alle Ergebnisse in die Hausarbeit.

Übungsklausur 1

1. (10 P) Geradeausprogramm: Programmieren Sie die Berechnung von K in der richtigen Reihenfolge und mit PC-gerechten Bezeichnungen (P, α und U) sind gegeben).

$$K = \sqrt{\frac{\cos(P) - K_\alpha e^{-U\alpha}}{\sqrt{\alpha^2 - \pi}}} \qquad \text{mit} \qquad K_\alpha = \frac{1 - e^{\alpha \cdot P}}{4\pi^2}$$

2. (20 P) Ablaufplan: Realisieren Sie in einem Programmstück mit if und else den folgenden Ablaufplan.



3. (20 P) Ein Vektor x sei mit n Zahlen x_1, ..., x_n gefüllt. Berechnen Sie mit Hilfe von for-Schleifen bitte die Spannweite, den Mittelwert und die Varianz.

Spannweite S = Maximum – Minimum $\quad \bar{x} = \left(\sum x_i\right)/n \qquad \sigma_n^2 = \dfrac{\sum_i^n (x_i - \bar{x})^2}{n}$

4. (20 P) Die Kosinusreihe läßt sich im Computer ohne Zahlenüberlauf berechnen, wenn $|x| \le \pi$ ist. Die Reihe lautet (siehe Mathe 1):

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!} + -... \text{ bis unendlich}$$

a) Wenn x>π dann subtrahiere in einer while-Schleife immer wieder 2π von x, bis die Bedingung $|x| \le \pi$ erfüllt ist. Die Funktion cos(x) hat Periode 2π und ändert dadurch ihren Wert nicht. Wenn dagegen der Fall x<–π vorliegt, dann addiere in einer while-Schleife immer wieder 2π zu x hinzu, bis die Bedingung $|x| \le \pi$ erfüllt ist.

b) Programmiere die Berechnung von cos(x), indem Variable NG (***N**ächstes **G**lied*) vor einer do-while-Schleife den Wert 1 bekommt, Schleifenzähler i=0 und Summenvariable KOSI=0 gesetzt werden. In der do-while-Schleife selbst wird zuerst das neue Glied NG zur Summe KOSI addiert, dann der Schleifenzähler um 1 erhöht und dann anschließend das neue Glied NG neu berechnet. Laut Formel ergibt sich das neue Glied, indem man seinen alten Wert einfach mit dem Wert des Bruches $-x^2 / ((2i)*(2i-1))$ multipliziert. Die do-while-Schleife wird beendet, falls $|NG| < 10^{-15}$ ist, d.h., der KOSI-Wert sich um weniger als 10^{-15} ändert.

5. (20 P) Klassendefinition: Definieren Sie nach den Vorlagen CNurEineZahl, CNurEinText eine Klasse CPerson, die Vorname und Alter in Jahren einer Person verwaltet. Zugriffsfunktionen sind *PutAlter, GetAlter, PutName, Getname* (alle inline). Stellen Sie im Programm OnDraw eine Person auf dem Bildschirm vor: Name und Alter, wobei Sie vor den Namen "StudentIn" und ein Blank setzen und das Alter um 3 Jahre verringern.

Übungsklausur 2

1. (5P) Wandeln Sie die Zahl 1387 zuerst in eine 12-stellige Dualzahl (Vornullen!) und dann in eine 4-stellige Hexadezimalzahl um.

2. (20 P) Einfaches Programm mit einer Verzweigung: Unter Nutzung der Iostream-Funktionen cin, cout, das die benötigten Werte γ, t, u anfordert, die Größe $\beta = \sqrt{e^{-\gamma \cdot t} + \pi - \cos(u)}$ berechnet und den Wert von β ausgibt. Ist der Radikant negativ, soll stattdessen die Ausschrift "Radikant negativ" erfolgen.

3. (40 P) Programm mit einer do-while und einer oder zwei for-Schleifen: Schreiben Sie ohne Nennung der Bibliotheken, aber evtl. mit Konstantendefinition, ein Programm:

- Ein Vektor DAT mit maximal N=200 Elementen soll unter Benutzung einer do-while-Schleife und der Funktionen cin und cout mit Messdaten gefüllt werden. Das Messreihenende ist erreicht, wenn eine Zahl mit einem Betrag $> 10^{15}$ eingegeben wird oder 200 Zahlen erreicht sind.
- Geben Sie mit printf und Formatstring die Anzahl n der eingetippten Zahlen aus
- Falls n>1 ist, berechnen Sie Mittelwert, Standardabweichung σ_{n-1} und die Spannweite Maximum-Minimum der Messdaten und geben die 3 Werte mit printf aus. Jede Zahl hat 25 Druckstellen, davon 8 Dezimalstellen.
- Falls n nicht größer als 1 ist, geben Sie mit printf die Warnung "Zu wenig Daten" aus.

4. (30 P) Tabelle mit fprintf auf eine Datei ausgeben: Schreiben Sie ohne Nennung der Bibliotheken, aber evtl. mit Konstantendefinition,

ein Programm, das eine Tabelle der Funktion $z(x) = \cos(2.9 \cdot x) + \sqrt{1 - e^{-x}}$ für x=0, 0.1, 0.2, ..., 20.0 berechnet und auf Datei ZFUNKTAB.DAT ausgibt.

- Schreiben Sie die Funktion als eigenständigen Programmteil vor dem main-Programm. Einziger Funktionsparameter ist x.
- Die Tabelle besteht aus einer zentrierten Überschrift "Tabelle z(x)", einer Leerzeile und einer Kopfzeile " x z(x)"
- Geben Sie x mit einer Dezimale, z(x) mit 8 Dezimalen aus.

5. (25 P) Funktionstabelle in das View-Fenster ausgeben in OnDraw:
Geben Sie in das View-Fenster dieselben Überschriften und Funktionswerte aus, die auch in Aufgabe 4 in der Konsolenanwendung gemacht werden.

Benutzen Sie pDC->TextOut(...) für die Textausgabe und sprintf(kette, format, liste) für die Herstellung einer Zeile, die Zahlen enthält.