

Von Pascal zu Delphi

Aufbau eines Delphi-Projektes

Aufbau einer Unit

Aufbau einer Prozedur

Grundlagen von ObjectPascal

Schleifen

Verzweigungen

Array

Prozeduren und Funktionen

Übungen

Ende

# Entwicklung der Programmiersprache PASCAL

1971 – Pascal (Wirth , Jensen – theoretische Definition der Sprache)

1981 – erste Implementation auf einem Rechner (Standard-Pascal)

1983 – Turbo-Pascal (Einheit von Editor, Compiler, Programmausführung)

1989 - Turbo-Pascal 5.5 mit TurboVision als Teil von Turbo-Pascal (Objektorientierte Programmierung)

1991 – Turbo-Pascal für Windows (FLOP)

1994 – Delphi 1.0

2001 – Delphi 6.0

Unter  
DOS

Unter  
Windows



# DELPHI

- Delphi ist eine **visuelle Programmierungsumgebung**, in der **objekt-** und **ereignisorientiert** gearbeitet wird.
- Die Sprache von Delphi ist **ObjectPascal**.
- Delphi stellt dem Entwickler die verschiedensten **Komponenten**, Dialoge, Funktionen und Prozeduren zur Verfügung, mit denen die Erstellung einer Programmoberfläche zum „Kinderspiel“ wird.

**Arbeits- und Sozialverhalten 1.1 - OBB 2000**  
Datei Übersichten Löschen About Hilfe

**Lehrer**   
**Klasse**

**Lerneinstellung**  
 a  b  c  d  e

**Zuverlässigkeit**  
 a  b  c  d  e

**Selbstständigkeit**  
 a  b  c  d  e

**Teamfähigkeit**  
 a  b  c  d  e

**Urteilsfähigkeit**  
 a  b  c  d  e

**LERNEINSTELLUNG** (Lernwille, Durchhaltevermögen, Leistungsbereitschaft, Fleiß, Konzentrationsfähigkeit, Mitarbeit, Aufmerksamkeit)  
.....

a) - zeigt außerordentlichen Lernwillen, Durchhaltevermögen und Leistungsbereitschaft

b) \_\_\_\_\_  
- arbeitet im Unterricht motiviert mit, beteiligt sich aktiv und kann sich über einen großen Zeitraum konzentrieren

c) \_\_\_\_\_  
- die gestellten Aufgaben werden meist motiviert und arbeitsfreudig ausgeführt  
- ist bereit, konzentriert zu arbeiten

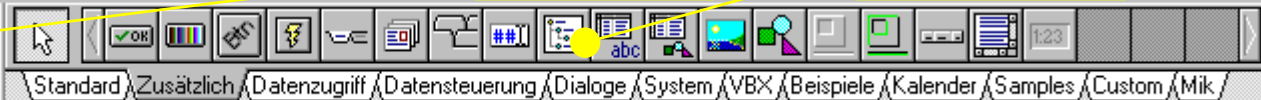
d) \_\_\_\_\_  
- gestellte Aufgaben werden noch ausreichend erfüllt  
- ist leicht ablenkbar

e) \_\_\_\_\_  
- erfüllt trotz Lernanreize wenig motiviert und selten die gestellten Aufgaben

Ihre Eintragungen werden gespeichert, wenn Sie den Schalter zur Wahl des nächsten Schülers betätigen.

Der Unterschied in der Arbeit mit z.B. TP und Delphi ist ungefähr der gleiche wie der zwischen dem Kaffee kochen mit Kaffeepulver, Wasser,... und dem Holen eines Kaffee vom Automaten.

**ABER:**  
Die inhaltliche Programmierung des Programmablaufs obliegt dem Nutzer, der dazu Pascal-Kenntnisse benötigt.



## Objektinspektor

Form1: TForm1

ActiveControl	
AutoScroll	True
+BorderIcons	[biSystemMenu,biV...
BorderStyle	bsSizeable
Caption	Form1
ClientHeight	86
ClientWidth	643
Color	clBtnFace
Ctl3D	True
Cursor	crDefault
Enabled	True
+Font	(TFont)
FormStyle	fsNormal
Height	113
HelpContext	0
Hint	
+HorzScrollBar	(TControlScrollBar)
Icon	(Leer)
KeyPreview	False
Left	192
Menu	
Name	Form1
ObjectMenuItem	
PixelsPerInch	96
PopupMenu	
Position	poDesigned
PrintScale	poProportional
Scaled	True
ShowHint	False
Tag	0
Top	97
+VertScrollBar	(TControlScrollBar)
Visible	False
Width	651
WindowMenu	
WindowState	wsNormal

## Form1

Dies ist das Formular, das der Nutzer dann sehen wird. Hier werden die Komponenten platziert.

## UNIT1.PAS

```

unit Unit1;

interface

uses
  SysUtils, WinTypes, WinProcs, Messages, Classes, Graphics, Contro
  Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private-Deklarationen }
  public
    { Public-Deklarationen }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.

```

Im Objectinspektor werden die Eigenschaften der Komponenten festgelegt und was bei bestimmten Ereignissen (z.B. Anklicken der Komponente) passieren soll.

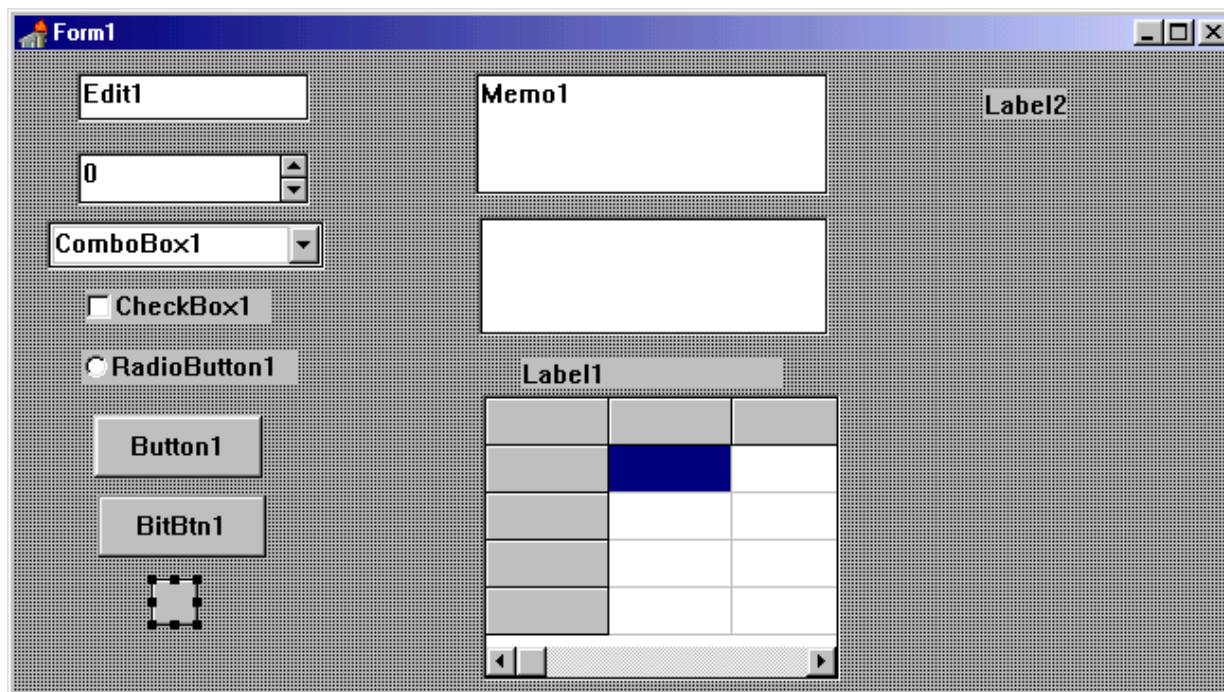
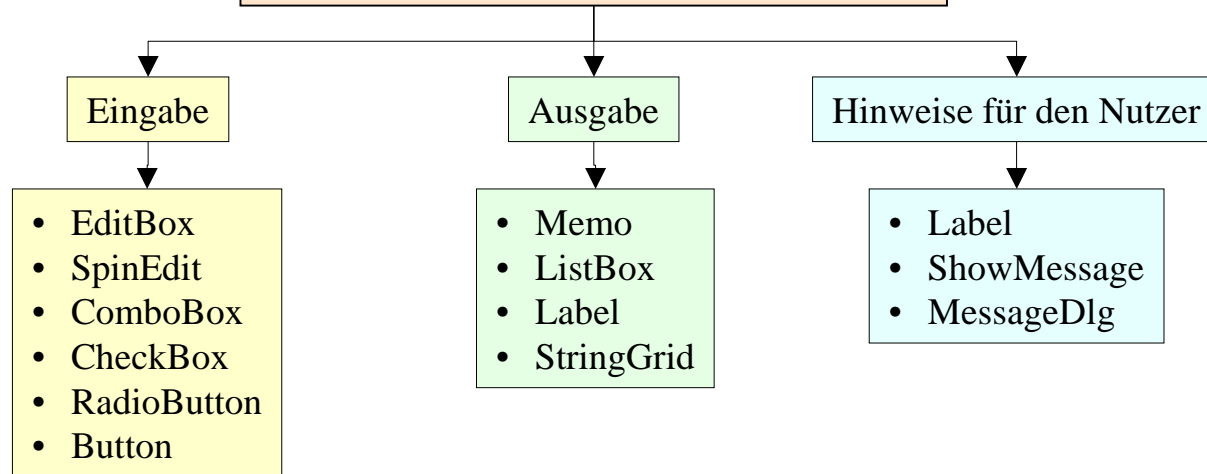
Editor

1: 1 | Geändert | Einfügen

Unit1

Eigenschaften | Ereignisse

# Grundlegende Komponenten in Delphi





## Komponenten zur Ausgabe:

- |              |  |
|--------------|--|
| · ListBox    | für Listen, Ausgabe in Spalten, Sortierung |
| · Memo       | für Texte                                  |
| · StringGrid | für tabellarische Ausgaben                 |
| · Label      | für Beschreibungen von Komponenten         |

Es können nur  
Texte (Strings)  
ausgegeben  
werden

---

## Komponenten zur Eingabe:

- |                 |                   |
|-----------------|-------------------|
| · EditBox, Memo | nur Text (String) |
| · SpinEdit      | nur LongInt       |

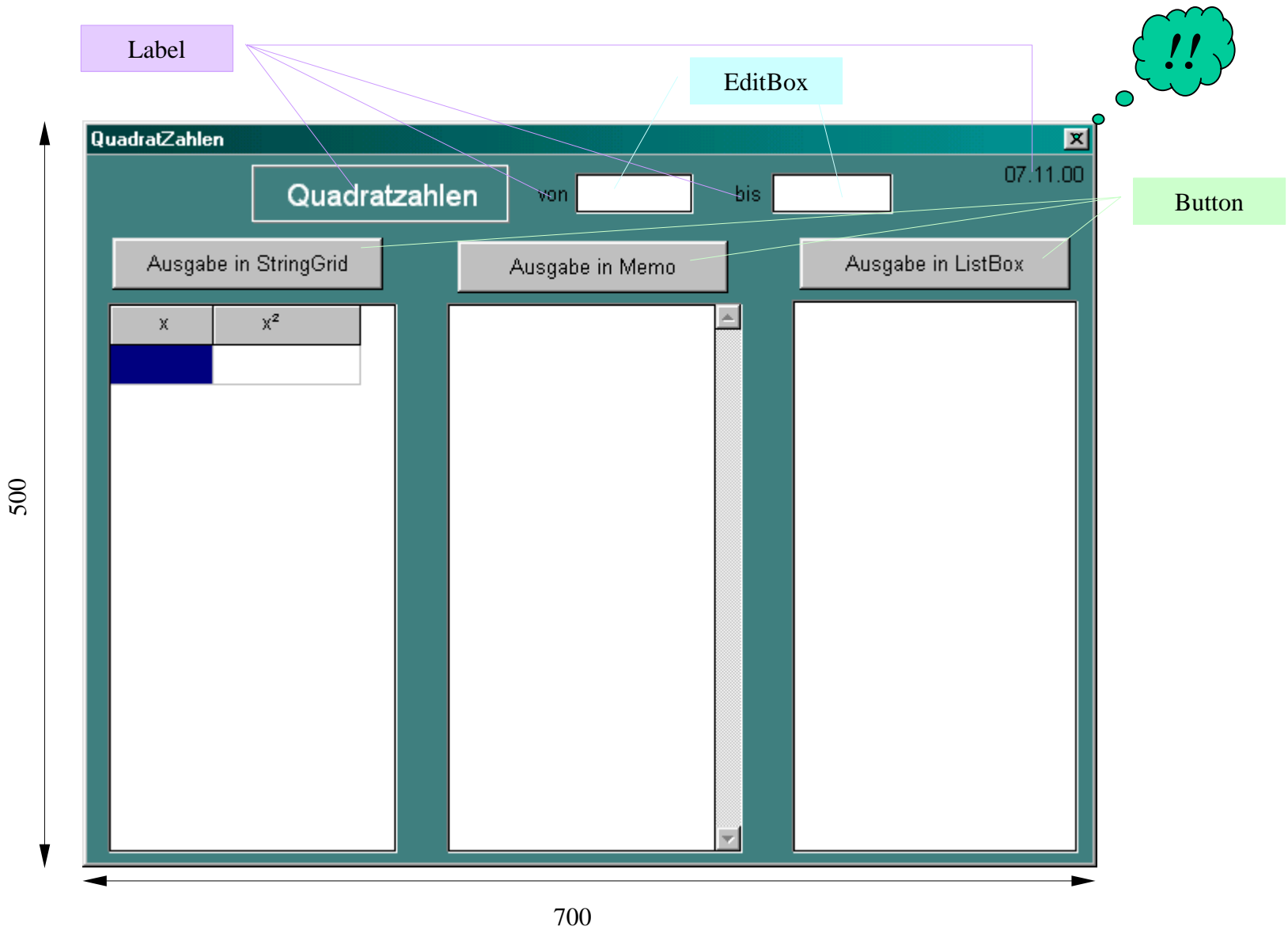
---

## Umwandlung von Zahlen in Strings u.u.

- |            |          |
|------------|----------|
| · IntToStr | Funktion |
| · StrToInt | Funktion |

Näheres zur  
Verwendung in der  
Delphi-Hilfe





Das Fenster soll auf dem Bildschirm stets zentriert erscheinen





# Aufbau eines Delphi-Projektes

Ein Delphi-Projekt besteht aus Dateien mit folgenden Suffixen:

- \*.dpr : die **D**elphi-**P**roject-Datei(en)
- \*.dfm : die **D**elphi-**F**ormular-Datei(en)
- \*.pas : die Quelltextdatei(en) der Unit(s) (Object**P**ascal)
- \*.res : in das Projekt einzubindende **R**essourcen (z.B. wav-Dateien, Icons, ...)
- \*.opt : **O**ptionen

☞ beim Compilieren entstehen \*.dcu-Dateien (**D**elphi-**C**ompiled **U**nit) und die \*.exe-Datei (**e**xecute)  
(Soll ein Delphi-Projekt an einem anderen Rechner weiterbearbeitet werden, genügt es, die Dateien mit den fünf o.g. Suffixen auf diesen Rechner zu bringen.)

Bis auf die Quelltextdatei (Unit) braucht der Programmierer sich um die anderen Dateien nicht zu kümmern - das erledigt Delphi.





# Aufbau einer Unit

```
unit _Test;
```

Der **UNIT-KOPF** besteht aus dem Schlüsselwort **unit** und dem Namen unter dem die Unit gespeichert wurde.

```
interface
```

```
uses
```

```
SysUtils, WinTypes, WinProcs,  
Messages, Classes, Graphics,  
Controls, Forms, Dialogs;
```

Die **USES-Anweisung** enthält die Bezeichner der Units, die Funktionen oder Prozeduren enthalten, die vom aktuellen Programm oder von der aktuellen Unit verwendet werden.  
In der Regel werden diese Eintragungen von Delphi vorgenommen.

```
type
```

```
THauptformular = class(TForm)  
  Schalter: TButton;  
  Textfeld: TMemo;  
  procedure SchalterClick(Sender:Tobject)  
private  
  { Private-Deklarationen }  
public  
  { Public-Deklarationen }  
end;
```

Im **TYPE**-Deklarationsteil werden die **globalen** Variablentypen definiert, die in ObjectPascal nicht vordefiniert sind. THauptformular ist also der Typ, den die Variable Hauptformular erhält.

Delphi ergänzt die Definition von THauptformular automatisch, wenn Komponenten auf das Formular gezogen werden und wenn für die Komponenten Ereignisbehandlungsroutinen geschrieben werden.

Interface-Teil

```
var
```

```
Hauptformular: THauptformular;
```

Im **VARIABLEN**-Deklarationsteil werden die **globalen** Variablen der Unit festgelegt.

```
implementation
```

```
{ $R *.DFM }
```

```
procedure THauptformular.SchalterClick(Sender: Tobject);
```

```
type TZahl=Real;
```

```
const c=3;
```

```
var Zahl : TZahl;
```

```
    i    : Byte;
```

```
begin
```

```
    zahl:=c;
```

```
    For i:=1 to 200 do zahl:=zahl+i*c;
```

```
end;
```

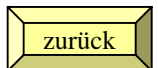
```
end.
```

Der **IMPLEMENTATION**-Teil einer Unit enthält den Programmteil der Prozeduren und Funktionen, die im Interface-

Teil der Unit deklariert wurden.



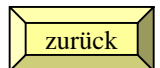
- Der **Interface-Teil** einer Unit bestimmt, was aus dieser Unit für Programme (oder andere Units), die diese Unit nutzen, sichtbar und zugänglich ist.
- Im Interface-Teil können Konstanten, Datentypen, Variablen, Prozeduren und Funktionen deklariert werden, die von anderen Programmen oder Units genutzt werden dürfen.
- Der Interface-Teil beginnt mit dem reservierten Wort `interface`, das nach dem Unit-Vorspann steht, und endet mit dem reservierten Wort `implementation`.





### einige vordefinierte Variablentypen in ObjectPascal

Bezeichner	Speicherbedarf		Bereich		
ShortInt	8 Bit		-128	...	127
Integer	16 Bit		-32768	...	32767
LongInt	32 Bit		-2147483648	...	2147483647
Word	16 Bit		0	...	65535
Byte	8 Bit		0	...	255
Boolean	1 Byte		false, true		
Char	1 Byte		ein Zeichen		
Real	6 Byte	Genauigkeit 11-12 Stellen	$2.9 \cdot 10^{-39}$	...	$1.7 \cdot 10^{38}$
Extended	10 Byte	Genauigkeit 19-20 Stellen	$3.4 \cdot 10^{-4932}$	...	$1.1 \cdot 10^{4932}$





- **Globale Variable:** kann (solange das Programm ausgeführt wird) von allen Prozeduren und Funktionen verwendet werden
- **Lokale Variable:** wird in einer Prozedur oder Funktion deklariert und steht deshalb nur solange diese Prozedur bzw. Funktion ausgeführt wird zur Verfügung.
- In einer Prozedur bzw. Funktion haben die lokal deklarierten Variablen Vorrang vor den global deklarierten, d.h., gibt es lokale und globale Variablen mit dem gleichen Bezeichner, dann wird innerhalb der Prozedur, in der die lokale Variable deklariert ist, der Wert der lokalen Variablen benutzt.

In einem Programm sollten nur so viele globale Variable verwendet werden wie unbedingt nötig, da sie den Speicher belasten. Außerdem wird bei Verwendung lokaler Variablen der Quelltext übersichtlicher.



# Aufbau einer Prozedur

Eine Prozedur wird von einer Prozedur-Anweisung aktiviert, die den Namen der Prozedur sowie die aktuellen Parameter, wenn vorhanden, angibt.

Für die unten stehende Prozedur also **SchalterClick (Self)**

```
procedure THauptformular.SchalterClick(Sender: TObject);
```

```
type TZahl=Real;
```

```
const c=3;
```

```
var Zahl : TZahl;
```

```
    i    : Byte;
```

```
begin
```

```
    zahl:=c;
```

```
    For i:=1 to 200 do zahl:=zahl+i*c;
```

```
end;
```

Im **Prozedurkopf** werden der Name der Prozedur und die formalen Parameter (falls vorhanden) angegeben.  
Für Ereignisbehandlungsprozeduren erstellt Delphi den Prozedurkopf automatisch.

Im **Deklarationsteil** der Prozedur werden die **lokalen** Typen (type), Konstanten (const) und Variablen (var) deklariert.  
Zu deren Deklaration wird der Bezeichner und der zuzuordnende Typ angegeben.

Im **Anweisungsteil** (zwischen begin und end) der Prozedur werden die beim Aufruf der Prozedur auszuführende Aktionen implementiert.

zurück

# Grundlagen von ObjectPascal

**Variablenbezeichner** beginnen mit einem Buchstaben und können eine Folge von Groß- bzw. Kleinbuchstaben, Ziffern und Zeichen enthalten, wobei einige Zeichen nicht vorkommen dürfen (Leerzeichen, Umlaute, \, /, (, ), ...). Zwischen Groß- und Kleinschreibung wird nicht unterschieden.

## Operatoren:

Zuweisungsoperator	:=		A:=1 ; a:=b
arithmetische Operatoren	+ ; - ; * ; /		A:= B+1 ; c:=2*c
	sqr(b)	=b <sup>2</sup>	a:= sqr(3)
	sqrt(b)	=√b	c:= sqrt(a)
	a DIV b	=[a/b]	c:= a div 4
	a MOD b	=a-(a DIV b)*b	x:= c MOD b
Vergleichsoperatoren	< ; > ; = ;		
	<= ; >=		
logische Operatoren	AND ; OR ;	und ; oder ;	
	NOT ; XOR	nicht ; entweder oder	

## Beachte:

die Variable, der ein Wert zugewiesen wird, muss einen Variablentyp haben, auf dem der zuzuweisende Wert abgelegt werden kann

Beispiele:

- für **a:=4/2** muss a mindestens vom Typ Real sein, da die Division stets RealZahlen ergibt
- für **b:=c mod 2** muss c einen ordinalen Typ haben, da die Operation MOD nur für ordinalen Typen definiert ist
- für **d:= a and 2** muss d vom Typ Boolean sein

Wenn dies beim Programmieren nicht beachtet wird, gibt Delphi die Fehlermeldung „Typen nicht vereinbar“ aus

Übung

# ÜBUNGEN

1. Entwickle in Delphi ein Programm, bei dem der Nutzer zwei natürliche Zahlen  $a$  und  $b$  eingeben kann. Es werden die Ergebnisse der vier Grundrechenarten ausgegeben, wobei für die Division von  $a$  und  $b$  das Ergebnis angegeben werden soll, wenn
  - auf drei Stellen nach dem Komma genau gerechnet wird
  - so wie in der 4.Klasse gerechnet wird (  $9 : 2 = 4 \text{ Rest } 1$  ).
2. Das Formular, das auf dem Bildschirm zentriert angezeigt wird, soll der Nutzer in der Größe nicht ändern können.
3. Entwickle in Delphi ein Programm, mit dem ein Nutzer zwei Brüche addieren kann. Das Ergebnis soll als nicht gekürzter Bruch ausgegeben werden.
4. Entwickle in Delphi ein Programm, mit dem ein Nutzer Volumen und Oberfläche eines Kreiszyllinders berechnen kann.

Trockentest

4. Entwickle in Delphi ein Programm, mit dem die Summe der natürlichen Zahlen von a bis b ( $a < b$ ) berechnet werden kann.

a) Unter ausschließlicher Verwendung von For-Schleifen,

b) Unter ausschließlicher Verwendung von Repeat-Schleifen,

c) Unter ausschließlicher Verwendung von While-Schleifen,

5. Welche Aussagen gelten bzgl. der gegenseitigen Ersetzbarkeit der Schleifen ?

6. Entwickle in Delphi zwei Programme (A und B), mit denen eine Wertetabelle für quadratische Funktionen  $f(x) = ax^2 + bx + c$  ( $a, b, c \in \mathbf{R}$ ) ausgegeben werden kann. Die Parameter a, b, c werden in EditBoxen eingegeben.

A) Start- und Endwert der Wertetabelle werden in SpinEdits festgelegt. Vom Startwert zum Endwert wird das Argument um 1 erhöht. In einem Memo werden das Argument und der zugehörige Funktionswert ausgegeben.

B) Start- und Endwert werden in EditBoxen festgelegt. Die Schrittweite zur Erhöhung des Arguments kann durch den Nutzer in einer EditBox beliebig festgelegt werden.




7. Entwickle in Delphi ein Programm, mit dem Zahlenpaare (a ; b) ausgegeben werden.

- a) a durchläuft die Werte von 1 bis 100 und b durchläuft die Werte von 51 bis 150 → also (1;51) , (2;52) , ... , (99;149) , (100,150)
- b) es werden alle Zahlenpaare (a;b) mit  $a, b \in \mathbf{N}$  und  $1 \leq a \leq 10$  und  $1 \leq b \leq 10$  ausgegeben
- c) es werden alle Zahlenpaare (a;b) mit  $a, b \in \mathbf{N}$  und  $1 \leq a \leq 10$  und  $1 \leq b \leq 10$  und  $a < b$  ausgegeben
- d) es werden alle Zahlenpaare (a;b) mit  $a, b \in \mathbf{N}$  und  $1 \leq a \leq 10$  und  $1 \leq b \leq 10$  und  $a > b$  ausgegeben

Lösungen

- I. Entwickle in Delphi ein Programm, mit dem geprüft werden kann, ob eine natürliche Zahl a Teiler einer natürlichen Zahl b ( $a < b$ ) ist.
- II. Ergänze das Programm zur Bruchrechnung so, dass das Ergebnis gekürzt wird (EUKLID'scher Algorithmus zur Bestimmung des g.g.T → siehe Tafelwerk)
- III. Entwickle in Delphi ein Programm, mit dem alle Teiler einer natürlichen Zahl bestimmt werden können.
- IV. Entwickle in Delphi ein Programm, mit dem geprüft werden kann, ob eine natürliche Zahl eine Primzahl ist.
- V. Entwickle in Delphi ein Programm, mit dem alle Primzahlen in einem vom Nutzer eingegebenen Bereich ausgegeben werden können.

Lösung C



A) Schreibe in Delphi ein Programm, mit dem getestet werden kann, welches Ergebnis man erhält, wenn aus einer beliebigen Zahl  $n$  mal ( $n \in \mathbf{N}$ ) die Quadratwurzel gezogen wird. Die Zwischenergebnisse sind auszugeben.

Lösung A

B) Die EULERSche Zahl  $e$  kann berechnet werden durch

$$e = 1 + \sum_{i=1}^{\infty} \frac{1}{i!} \text{ mit } i \in \mathbf{N} \text{ und } i! = \prod_{j=1}^i j = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (i-2) \cdot (i-1) \cdot i$$

Lösung B

B1) Schreibe ein in Delphi ein Programm, mit dem 30 Näherungswerte für  $e$  berechnet werden.

B2) Schreibe ein in Delphi ein Programm, mit dem solange Näherungswerte für  $e$  berechnet werden, bis sich zwei aufeinanderfolgende Näherungswerte um weniger als  $10^{-5}$  unterscheiden.

C) Schreibe in Delphi ein Programm zur Ermittlung der Quersumme einer Zahl.

Lösung C



Als es noch keine Rechenhilfsmittel (Rechenstab, Taschenrechner,...) gab, mussten diejenigen, die Werte für z.B.  $\sin(x)$  benötigten, diese aus Tafeln ablesen. Diese Tafeln musste jedoch auch erst einmal jemand berechnet haben.

Dazu wurde die TAYLOR-Entwicklung genutzt.

Für  $f(x)=\sin(x)$  lautet die TAYLOR-Entwicklung:

$$\sin(x) = \sum_{i=0}^{\infty} (-1)^i \cdot \frac{x^{2i+1}}{i!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \dots$$

$$\text{mit } i! = \prod_{j=1}^i j = 1 \cdot 2 \cdot 3 \cdot \dots \cdot (i-1) \cdot i$$

Will man also z.B.  $\sin(\pi/4)$  ausrechnen, so setzt man  $\pi/4$  in die Formel ein und kann den Wert so beliebig genau berechnen.

Schreibe ein Programm, mit dem der Nutzer für beliebiges  $x$   $\sin(x)$  berechnen lassen kann. Der Nutzer soll die von ihm gewünschte Genauigkeit (z.B. 0,0000001) eingeben können.



# Trockentest

Auf einem Formular befinden sich ein Memo und ein Schalter. Nach dem Anklicken des Schalters wird die folgende Prozedur abgearbeitet. Gib an, was im Memo ausgegeben wird.



```
procedure TForm1.SchalterClick(Sender:Tobject);
const z=100;
var a, b, i : Integer;
      x      : Real;
begin
  Memo.clear;
  a:=1;
  for i:=1 to 10 do a:=2*a;
  memo.Lines.Add('a= '+IntToStr(a));
```

```

x:=1;
Repeat
  x:=x/2;
Until x<1e-2;
memo.Lines.Add('x= '+FloatToStr(x));
```

```

While x<3 do x:=2*x;
memo.Lines.Add('x= '+FloatToStr(x));
```

```

i:=0; a:=0;
Repeat
  inc(i); {i:=i+1}
  a:=a+i;
Until i=z;
memo.Lines.Add('a= '+IntToStr(a));
```

```

a:=0; b:=1;
Repeat
  inc(a);
  x:=x+a*b;
Until x=z;
memo.Lines.Add('x= '+FloatToStr(x));
end;
```

Lösung

# Schleifen

**Problem:** Es soll die Summe  $\sum_{i=a}^b i$  ( $a, b, i \in \mathbf{N}$ ;  $a < b$ ) berechnet werden.

Schleifen ermöglichen es, eine oder mehrere Anweisungen wiederholt auszuführen, solange oder bis eine Bedingung erfüllt ist.

Es gibt drei Arten von Schleifen:

1. For ... to / DownTo
2. Repeat ... Until
3. While ... Do

Schleife	Verwenden, wenn
for	die Anzahl der Schleifendurchläufe genau bekannt ist.
Repeat ... until	die Schleife mindestens einmal ausgeführt werden soll/muss/kann, bevor die Bedingung getestet wird.
While ... do	die Bedingung vor Eintritt in die Schleife geprüft werden soll/muss.

## Syntax:

**For** Laufvariable:=Startwert **to** Endwert **do**

```
begin
  {Anweisungen}
End;
```

- Laufvariable, Startwert und Endwert müssen einen ordinalen Typ haben
- Begin und end können entfallen, wenn nur eine Anweisung ausgeführt werden soll

**Repeat**

```
{Anweisungen}
```

**Until** Bedingung;

- Nachdem die Anweisungen einmal ausgeführt wurden, wird die Bedingung geprüft
- Ist sie erfüllt, wird die Schleife verlassen
- Ansonsten werden die Anweisungen ausgeführt, bis die Bedingung erfüllt ist

**While** Bedingung **do**

```
begin
  {Anweisungen}
end;
```

- zuerst wird die Bedingung geprüft
- solange sie erfüllt ist, werden die Anweisungen ausgeführt
- Begin und end können entfallen, wenn nur eine Anweisung ausgeführt werden soll

```
Var a,b,i,summe : LongInt;
```

```
BEGIN
```

```
  Summe:=0;
```

```
  For i:=a to b do summe:=summe+i;
```

```
END;
```

```
Var a,b,i,summe : LongInt;
```

```
BEGIN
```

```
  Summe:=0; i:=a;
```

```
  Repeat
```

```
    Summe:=summe+i;
```

```
    i:=i+1;
```

```
  Until i=b;
```

```
END;
```

```
Var a,b,i,summe : LongInt;
```

```
BEGIN
```

```
  Summe:=0; i:=a;
```

```
  While i<=b do
```

```
    begin
```

```
      Summe:=summe+i;
```

```
      i:=i+1;
```

```
    end;
```

```
END;
```

Übung

Trockentest

Bedingungen

# Bedingungen

Eine Bedingung ist erfüllt (wahr) oder nicht erfüllt (falsch). Das Ergebnis ist also vom Typ Boolean.

Eine Bedingung kann sein:

Ein Vergleich	$A=1 ; a<1 ; a>b ; a<=b$	
Eine Verknüpfung zweier Teilaussagen durch logische Verknüpfungen (Junktoren)	$A \text{ and } B ; A \text{ or } B ; A \text{ and } (B \text{ or } C) ; (A \text{ and } B) \text{ xor } (A \text{ or } B)$	die Teilaussagen müssen ein Ergebnis vom Typ Boolean liefern

Welche Ergebnisse die Junktoren liefern, wird in einer Wahrheitstabelle dargestellt:

		Negation	Konjunktion	Disjunktion	Antivalenz
		Nicht (non)	Und (et)	Oder (vel)	Entweder ... Oder (aut)
		Not	And	Or	Xor
P	Q	$\neg P$	$P \wedge Q$	$P \vee Q$	$P \oplus Q$
w	w	f	w	w	f
w	f	f	f	w	w
f	w	w	f	w	w
f	f	w	f	f	f

Beispiel:

1. Die Bedingung  $(a>2) \wedge (a<5)$  ist wahr für ...
2. Die Bedingung  $(a>2) \vee (a<5)$  ist wahr für ...

# Verzweigungen

Verzweigungen ermöglichen es, die Programmausführung in Abhängigkeit von der Gültigkeit einer Bedingung in verschiedener Weise fortzusetzen.

<b>If</b> Bedingung <b>then</b>	Wenn Bedingung erfüllt dann	If a mod 2=0 then
<b>Begin</b>		Begin
Anweisungen	Tue dies	Labell.caption:=„gerade Zahl“
<b>End</b>		End
<b>Else</b>	Ansonsten	Else
<b>Begin</b>		Begin
Anweisungen	Tue das	Labell.caption:=„ungerade Zahl“
<b>End;</b>		End;

- Folgt auf then bzw. else nur eine Anweisung können begin und end weggelassen werden
- **ACHTUNG: vor else darf kein Semikolon stehen**

Mehrere Verzweigungen können zu

einer Auswahl zusammengefasst werden.

```
If a=1 then a:=2*a;
If a=2 then a:=3*a;
If a>2 then a:=a div 3;
```

```
Case a of
  1: a:=2*a;
  2: a:=3*a
End
Else a:=a div 3;
```

- Der Selector a muss einen ordinalen Typ haben
- Bereiche werden durch Punkte (..) oder Komma (,) getrennt.

```
Byte, Word,
Integer,
Char
Case a of
  1 :
  2,3 :
  4..10 :
End;
```

Bedingungen

Übung

Trockentest



```
procedure TForm1.Button1Click(Sender: TObject);
var a,b,c,d,i,j,k,l,z :Integer;
begin
  a:=0;
  for i:=1 to 10 do
    if i mod 2=1 then inc(a);
  Mem1.lines.add('a='+inttostr(a));
  k:=-5; b:=0;
  Repeat
    if k<0 then k:=k+2 else dec(k);
    inc(b);
  until k=0;
  Mem1.lines.add('b='+inttostr(b));
  k:=8; c:=0;
  for i:=1 to 8 do
    if k mod i=0 then c:=c+i;
  Mem1.lines.add('c='+inttostr(c));
  l:=23454; d:=0;
  Repeat
    z:=l mod 10;
    d:=d+z;
    l:=l div 10;
  until l=0;
  Mem1.lines.add('d='+inttostr(d));
end;
```



**Arrays** sind ein- oder mehrdimensionale Bereiche, die **viele Variablen des gleichen Typs** enthalten. Auf jede Variable im Array kann über den **Namen** des Arrays **und** den **Index** der Variable zugegriffen werden, der in Klammern angegeben wird.

Bsp.:

**VAR z : Array[1..20] of LongInt**

- Deklariert das eindimensionale Array z mit den Indizes von 1 bis 20 vom Typ LongInt
- z besteht also aus 20 Variablen die alle den Typ LongInt haben
- mit *For i:=1 to 20 do z[i]:=Sqr(i)-1* kann z belegt werden,
- mit *For i:=1 to 20 do ListBox1.Lines.add('z['+IntToStr(i)+''] = '+IntToStr(z[i]))* kann z in einer ListBox ausgegeben werden

**VAR z : Array[1..20,5..10] of Byte**

- Deklariert das zweidimensionale Array z vom Typ Byte
- z besteht also aus  $20*6=120$  Variablen die alle den Typ Byte haben
- mit *For i:=1 to 20 do*  
    *for j:=5 to 10 do z[i,j]:=i\*j*  
kann z belegt werden,
- mit *For i:=1 to 20 do*  
    *For j:=5 to 10 do ListBox1.Lines.add('z['+IntToStr(i)+''];'+IntToStr(j)+'']='+IntToStr(z[i,j]))*  
kann z in einer ListBox ausgegeben werden

**Problem:** Der Computer soll einen Würfel simulieren. Es soll n-mal gewürfelt werden und nach Durchführung aller Versuche werden die absoluten und relativen Häufigkeiten ausgegeben.

### Vorüberlegungen (Analyse) :

- die Anzahl der Versuche steht fest → For-Schleife
- es muss während des Würfeln gezählt werden, wie oft jede der möglichen Zahlen (1..6) vorkommt
- für jede der möglichen Zahlen passiert dasselbe: Wenn die Zahl x gewürfelt wurde, dann erhöhe den zugehörigen Zähler
- nach Ende der For-Schleife müssen alle Ergebnisse zur Verfügung stehen
- es bietet sich also ein Array mit den Indizes von 1...6 an

```
For v:=1 to ... do begin
    x:= ... ;
    Inc(z[x]);
end;
```

Programm



Strings



In ObjectPascal ist Variablentyp **STRING** vordefiniert.

Dieser stellt quasi ein Array of Char dar und wird syntaktisch auch so behandelt.

- `Var s :String` → deklariert eine Variable s vom Typ String (maximal 255 beliebige Zeichen)
- `Var s :String[7]` → deklariert eine Variable s vom Typ String (maximal 7 beliebige Zeichen)
- `s:=Edit1.Text` → die Eigenschaft Text einer Komponente Edit1 wird auf s abgelegt
- `Label1.Caption:=s` → der Inhalt von s wird auf der Eigenschaft Caption eines Labels abgelegt
- `l:=Length(s)` → auf l wird die Anzahl der Zeichen von s abgelegt
- `For z:=1 to Length(s) do b:=s[z]` → auf b werden nacheinander alle Zeichen von s abgelegt
- `b:='liege' ; c:='F' ; x:=c+b` → Strings können „addiert“ werden; auf x steht Fliege
- weitere Befehle zur Arbeit mit Strings finden sich in der Delphi-Hilfe (copy, concat, ...)

→ Programm: Buchstaben wandern lassen

```
Procedure Pause(l:Word);
Var start :LongInt;
Begin
  start:=GetTickCount;
  Repeat
    Application.ProcessMessages
  Until GetTickCount-start>=l
End;
```

*Aufruf mit z.B. PAUSE(10)*



Zur Arbeit mit dem Inhalt einer Komponente Memo

- der Inhalt eines Memos steht auf der Eigenschaft Lines
- alle Lines eines Memos bilden ein Array, d.h. mit `s:=Memo1.Lines[3]` wird die 4.Zeile eines Memos auf s abgelegt
- jede Zeile eines Memos ist für sich ein String
- die Anzahl der Zeilen eines Memos stellt man mit `Memo1.Lines.Count` fest

→ Programm: Buchstaben zählen

