

# Turbo-Pascal - 1. Halbjahr

<http://worgtsone.scienceontheweb.net/worgtsone/> - <mailto:worgtsone@hush.com>

01. September 2003 - 13. Oktober 2011

## Inhaltsverzeichnis

<b>1</b>	<b>Allgemeines</b>	<b>6</b>
1.1	Programmiersprache . . . . .	6
1.2	Wertschöpfungskette . . . . .	6
1.3	Compiler vs. Interpreter . . . . .	7
<b>2</b>	<b>Beurteilung von Leistungen</b>	<b>7</b>
<b>3</b>	<b>Hello World + WRITELN</b>	<b>8</b>
3.1	Problem . . . . .	8
3.2	Struktogramm . . . . .	8
3.3	Syntax . . . . .	8
3.4	Programm Version 0.9 (ungenügend) . . . . .	8
3.5	Programm Version 0.99 - o.k. . . . .	8
3.6	READLN; . . . . .	9
3.6.1	Strings (Zeichenketten) . . . . .	9
3.6.2	Zahlen und Rechenaufgaben . . . . .	9
3.6.3	Listen . . . . .	9
3.6.4	Formatierung . . . . .	10
3.7	Übung . . . . .	11
3.7.1	Struktogramm . . . . .	11
3.7.2	Ausgabe . . . . .	11
3.7.3	Rechenaufgaben . . . . .	11
<b>4</b>	<b>Variablen</b>	<b>12</b>
4.1	Datentypen . . . . .	12
4.1.1	Definieren im Programmkopf . . . . .	12
4.1.2	Zuweisen im Hauptprogramm . . . . .	12
4.1.3	Schreibtischtest . . . . .	12
4.1.4	Zuweisung im Struktogramm . . . . .	13
4.2	Deutsche Umlaute und Division . . . . .	13
4.3	Ausgabe . . . . .	13
4.3.1	Ausgabe im Struktogramm . . . . .	13
4.4	Übung . . . . .	13
4.5	Einlesen von Variablen von der Tastatur . . . . .	14
4.6	Übung . . . . .	15

4.6.1	Hallo . . . . .	15
4.6.2	Zwei Zahlen . . . . .	15
4.6.3	Würfel . . . . .	15
4.6.4	Brutto . . . . .	15
4.6.5	Quader1 . . . . .	15
4.6.6	Prozent-Anteil . . . . .	15
4.6.7	Kreis . . . . .	15
<b>5</b>	<b>Funktionen - eingebaut</b>	<b>16</b>
5.1	Unäre Funktionen (nur ein Argument) . . . . .	16
5.2	Binäre Funktionen (Zwei Argumente) . . . . .	16
5.3	Die Konstante PI . . . . .	16
5.4	Übung . . . . .	17
5.4.1	Beispiele . . . . .	17
5.4.2	wuerfel2 . . . . .	17
5.4.3	quader1 . . . . .	17
5.4.4	prozent1 . . . . .	17
5.4.5	math1 . . . . .	17
5.4.6	Benzinverbrauch . . . . .	17
5.4.7	quadrgl1 . . . . .	17
<b>6</b>	<b>Bedingte Verzweigung</b>	<b>18</b>
6.1	Einfaches Beispiel: Regen . . . . .	18
6.2	Verschachteltes Beispiel: Regen und Schirm . . . . .	18
6.3	Bedingung . . . . .	19
6.4	Syntax . . . . .	19
6.5	AND, OR und NOT . . . . .	20
6.6	Musterlösungen der ersten drei Übungen . . . . .	21
6.7	Übung . . . . .	22
6.7.1	Zahlenterror . . . . .	22
6.7.2	Gehalt . . . . .	22
6.7.3	Namen . . . . .	22
6.7.4	Bauspar . . . . .	22
6.7.5	Bank . . . . .	22
6.7.6	QuadrGl2 . . . . .	22
6.7.7	DreiName . . . . .	22
6.7.8	Geradengleichung . . . . .	22
<b>7</b>	<b>Wiederholung 1</b>	<b>23</b>
<b>8</b>	<b>Zählschleife</b>	<b>25</b>
8.1	Syntax . . . . .	25
8.2	Übung . . . . .	26
8.2.1	Zahlen zusammenzählen . . . . .	26
8.2.2	FORDO01 . . . . .	26
8.2.3	FORDO02 . . . . .	26

8.2.4	Fakultät . . . . .	26
8.2.5	Großes Einmaleins . . . . .	26
8.2.6	ASCII-Tabelle . . . . .	26
8.2.7	SINUS-Verlauf . . . . .	26
8.2.8	WasIstDas . . . . .	26
8.2.9	Einfacher . . . . .	26
<b>9</b>	<b>Unit CRT</b>	<b>27</b>
9.1	Units . . . . .	27
9.2	Probleme mit mehr als 100 MHz . . . . .	27
9.3	More Information . . . . .	27
9.4	CRT-Aufrufe, die wir nie benutzen . . . . .	27
9.5	CRT-Aufrufe, die wir selten benutzen . . . . .	28
9.6	Übung CRT . . . . .	28
9.6.1	Bunt . . . . .	28
9.6.2	Würfel3 . . . . .	28
9.6.3	Quader3 . . . . .	28
<b>10</b>	<b>Schleifen mit Start- oder Stopbedingung</b>	<b>29</b>
10.1	Struktogramm . . . . .	29
10.2	Syntax . . . . .	29
10.3	Übung . . . . .	30
10.3.1	Summe der $1/2^n$ . . . . .	30
10.3.2	Menüführung . . . . .	30
10.3.3	Simulation . . . . .	30
10.3.4	Barbie's Primzahltest . . . . .	30
10.3.5	Ken's Quader . . . . .	30
10.3.6	Summe aller $1/x^2$ . . . . .	31
10.3.7	Iteratives Wurzelziehen nach Hans . . . . .	31
10.3.8	Iteratives Wurzelziehen nach Lisa . . . . .	31
10.3.9	Diophant . . . . .	31
<b>11</b>	<b>Wrapup Schleifen und Random [optional]</b>	<b>32</b>
11.1	Wrapup Schleifen . . . . .	32
11.2	Random . . . . .	32
11.2.1	Übung Würfel1 . . . . .	32
11.2.2	Übung Würfel2 . . . . .	32
<b>12</b>	<b>Bedingte mehrfache Verzweigung</b>	<b>33</b>
12.1	Syntax . . . . .	33
12.2	Übungen . . . . .	34
12.2.1	CASE01 . . . . .	34
12.2.2	CASE02 . . . . .	34
12.2.3	Taschenrechner . . . . .	34
12.2.4	Wissenschaftlicher Taschenrechner . . . . .	34
12.2.5	Buchung2 . . . . .	34

<b>13 Wiederholung 2</b>	<b>35</b>
<b>14 Funktionen und Prozeduren - sauber</b>	<b>37</b>
14.1 Funktionen - selbstgeschrieben . . . . .	37
14.1.1 Syntax . . . . .	37
14.2 Prozeduren . . . . .	38
14.2.1 Modularisierung . . . . .	38
14.3 Übung . . . . .	39
14.3.1 Teste zwei INTEGER auf Gleichheit . . . . .	39
14.3.2 Minimum von 5 INTEGER . . . . .	39
14.3.3 Wertetabelle . . . . .	39
14.3.4 Rekursiv: Fakultät . . . . .	39
14.3.5 Größter Gemeinsamer Teiler . . . . .	39
14.3.6 BarbieMod . . . . .	39
14.3.7 Quader4 in Prozeduren zerlegen . . . . .	39
14.3.8 Unlösbare Aufgabe - Newton- und Sekanten-Verfahren . . . . .	40
14.3.9 Einstein . . . . .	40
<b>15 Funktionen und Prozeduren - unsauber</b>	<b>41</b>
15.1 Call by Value und Call by Reference . . . . .	41
15.2 Bewertung des Call by Reference . . . . .	41
15.2.1 Übung: procedures tausche . . . . .	41
15.3 Wie funktioniert Call by Value? [optional] . . . . .	42
15.4 Wie funktioniert Call by Reference? [optional] . . . . .	42
15.5 Didaktische Bewertung der Calls by XXX bzw. Global-Lokal . . . . .	42
15.6 Globale und lokale Variablen . . . . .	43
<b>16 Zufallszahlen und Konstanten</b>	<b>44</b>
16.1 Zufallszahlen . . . . .	44
16.2 Konstanten . . . . .	44
<b>17 Felder</b>	<b>45</b>
17.1 Übung . . . . .	47
17.1.1 Mein erstes Feld . . . . .	47
17.1.2 Summe, Minimum, Mittelwert, Maximum . . . . .	47
17.1.3 Lottozahlen . . . . .	47
17.1.4 Bubblesort . . . . .	47
17.1.5 MinSort . . . . .	47

### **Disclaimer**

Wissen ist zum Teilen da. Ich teile mein Wissen mit Ihnen, lieber Kollege.

Ich bin aber nicht perfekt. Unter [worgtsone@hush.com](mailto:worgtsone@hush.com)  
nehme ich dankbar Ihre Verbesserungsvorschläge entgegen.

\*

**Legal Blurb:** Alle Informationen in diesem Dokument sind falsch, unvollständig,  
irreführend, irrelevant und / oder funktionieren einfach nicht.

Wenn Sie es trotzdem benutzen, und es geht dabei etwas kaputt, ist das Ihr  
Problem, nicht meins.

\*

**Bitte teilen Sie meine Web-Adresse nicht Ihren Schülern mit.**

# 1 Allgemeines

## 1.1 Programmiersprache

Pascal ist eine höhere Programmiersprache.

**Eine Programmiersprache ist ein Werkzeug zur Kodierung von Algorithmen in menschen-lesbarer, compilierbarer Form.**

Eine höhere Programmiersprache beinhaltet Konstrukte wie zB Kontroll-Strukturen, Datentypen, etc. Sie sind viel komplexer als die Befehle, die die CPU direkt ausführen kann.

Je höher die Programmiersprache ist, desto weniger Routineaufgaben muß der Programmierer selber kodieren. Dafür muß er aber die Konstrukte kennen und richtig anwenden können.

**Mächtige Konstrukte beinhalten stets auch die Möglichkeit, mächtig große Fehler zu machen.)**

## 1.2 Wertschöpfungskette

Am Anfang steht das *Problem*, meistens eine langweilige Routineaufgabe.

In der Wirtschaft steht am Anfang der Kunde, der eine Verringerung von Routineaufgaben = Entlastung haben will. Er mißt die angebotenen Lösungen anhand von Funktion, Bedienerfreundlichkeit, Preis, Wartungsaufwand etc.

Das Problem müssen wir erstmal verstehen, deshalb wird es *analysiert*.

**Analyse ist das Aufteilen eines größeren Problems in mehrere, leicht lösbare Einzelprobleme.**

Die Problemlösung entwerfen wir in einem *Algorithmus*.

**Ein Algorithmus ist eine kleinschrittige Beschreibung, wie man ein Problem löst. Ein Algorithmus ist an keine Sprache gebunden. Man könnte sogar Deutsch nehmen.**

Damit wir dabei nicht so viel zu schreiben brauchen und es trotzdem jeder versteht, benutzen wir *Struktogramme*.

Der Algorithmus wird dann getestet und in einer Programmiersprache kodiert. Wir benutzen Pascal, weil sie

1. hoch, aber nicht zu hoch ist und
2. zum Strukturierten Programmieren zwingt.<sup>1</sup>

---

<sup>1</sup>Objektorientierung kommt später.

### 1.3 Compiler vs. Interpreter

Kein Rechner kann Pascal-Programme direkt ausführen (nur Maschinencode).

Wir benötigen ein Programm, das menschen-lesbares Pascal in maschinen-ausführbare Maschinensprache übersetzt. Dieses Programm heißt Compiler oder Interpreter.

**Ein Compiler nimmt das *gesamte* Pascal-Programm und erzeugt dazu eine ausführbare, binäre (nicht menschen-lesbare) Datei.**

Vorteile:

- Alle Pascal-Befehle werden nur einmal übersetzt. Während der Programm-Ausführung muß nichts mehr übersetzt werden. Compilierte Programme gelten als schnell.
- Das compilierte Programm ist auf dem OS in der Rechnerarchitektur, in der es erstellt wurde, verteilbar (läuft überall);
- Keiner kann hineingucken, welchen Algorithmus es verwendet, oder wie schlecht es geschrieben ist.

*Nachteil:* Bei Syntaxfehler im Programm erzeugt der Compiler Fehlermeldung und bricht die Compilierung ab. Dann muß man den Quelltext nachbessern, bis er durchläuft.

**Ein Interpreter nimmt die erste Zeile und macht das, was darinsteht. Dann die zweite, usw. bis Ende.**

*Vorteil:* Was o.k. ist, wird auch ausgeführt.

*Nachteil:* Viele Zeilen werden mehrfach durchlaufen und müssen deshalb wieder und wieder übersetzt werden. Interpreter gelten als langsam.

## 2 Beurteilung von Leistungen

Schülerleistungen sind

- mündliche Leistungen (Anwesenheit, Aufmerksamkeit, weiterführende Fragen und Ideen, Störgrad);
- Mappenführung;
- Projekte und Referate;
- Klausuren.
  1. Handschriftlich auf Papier. Nicht am Rechner.
  2. Hilfsmittel: Stift, Papier, Taschenrechner, Lineal. Basta.
  3. Übliche Aufgaben umfassen:
    - Vom Problem zum Struktogramm, und rückwärts.
    - Vom Struktogramm zum Programm, und rückwärts.
    - Beides, und rückwärts.

### 3 Hello World + WRITELN

#### 3.1 Problem

Das Programm soll "Hello World!!!" ausgeben.

#### 3.2 Struktogramm

```
Ausgabe ('Hello World!!!');
```

Ein Struktogramm besteht aus lauter Kästchen, die von oben nach unten durchlaufen werden.

#### 3.3 Syntax

```
writeln(...); gibt die Liste in den Klammern aus und anschließend ein CR-LF.
write(...); gibt die Liste in den Klammern aus. Sonst nichts.
Falls die Liste leer ist, muß man die Klammern weglassen.
Listentrennzeichen ist das Komma (",").
```

CR-LF ist DOS für "NeueZeile". CR ist carriage return, Wagenrücklauf (als ob man den Wagen einer Schreibmaschine nach rechts schiebt). LF ist line feed, Zeilenvorschub (als ob man an einer Schreibmaschine den Zeilenschalthebel betätigt).

#### 3.4 Programm Version 0.9 (ungenügend)

```
PROGRAM p;

BEGIN
  WRITELN ('Hello World!!!');
END.
```

Ich schreibe Pascal-Schlüsselwörter GROSS, den Rest klein.

**Jedes Programm hat einen Kopf (hier: mit Programmname) und einen Anweisungsteil, der zwischen begin und end eingeklammert ist.**

#### 3.5 Programm Version 0.99 - o.k.

```
PROGRAM helloworld;          (* für putty *)

BEGIN
  WRITELN ('HelloWorld von GRM, 22.10.2001: sagt Hello World.');
```

```
  WRITELN ('Hello World!!!');
```

```
END.
```

Ich verlange:

- einen "sprechenden" Programmnamen.
- "sprechende" Namen für Variablen und Funktionen.
- eine Zeile mit der Info "WieWerWannWas".
- eine Ausgabe, wenn der User etwas eingeben soll.

Pascal verbietet

- Leerzeichen in Namen (= "Bezeichner");
- mehrfach denselben Namen für Programm, Variablen, Funktionen, Prozeduren, etc.

```
PROGRAM helloworld;           (* für Turbo-IDE *)

BEGIN
  WRITELN ('HelloWorld von GRM, 22.10.2001: sagt Hello World. ');
  WRITELN ('Hello World!!!');
  WRITELN ('CR to continue...');
  READLN;
END.
```

### 3.6 READLN;

Wartet auf Drücken der CR-Taste.

#### 3.6.1 Strings (Zeichenketten)

Zeichenketten werden in Hochkommas gesetzt.

In die Klammern darf man Zeichenketten schreiben - die gibt der Rechner dann wörtlich aus. Pascal beachtet dabei sogar GROSS- und kleinschreibung.

Pascal kann deutsche Umlaute nur auf Umwegen.

Ein Hochkomma innerhalb einer Zeichenkette wird durch ein doppeltes Hochkomma dargestellt.

#### 3.6.2 Zahlen und Rechenaufgaben

In die Klammern darf man auch Rechenaufgaben schreiben, zB  $2+3*4$ .

Pascal kennt die Regel: Punktrechnung kommt vor Strichrechnung.

Rechenaufgaben mit Divisionen geben ein merkwürdiges Zahlenformat.

Pascal verwendet den Dezimalpunkt, nicht Dezimalkomma.

"Rechenaufgaben" heißen unter Informatikern auch "Ausdrücke". Sie rechnen mit Zahlen, Zeichenketten, Vektoren, Matrizen, Tabellen und aberwitzigen Verknüpfungszeichen... aber nicht heute.

#### 3.6.3 Listen

In den Klammern darf man mehrere Einträge hintereinander schreiben, wenn sie durch Kommas getrennt werden.

Man darf in der Liste sogar Zeichenketten und Rechenaufgaben mischen.

### 3.6.4 Formatierung

Hinter jeden Listeneintrag darf man ':int' schreiben. int ist dabei eine Rechenaufgabe ohne Nachkommastellen.  
Der Eintrag wird dann RECHTSBÜNDIG in ein Textfeld der Breite int ausgegeben.

Hinter Listeneinträge vom Typ `REAL`, d.h mit Nachkommastellen, darf man NOCH EIN :int schreiben. Das bewirkt:

1. Die Darstellung wird von exponentieller auf "normale" Schreibweise umgeschaltet.
2. int gibt die Anzahl der Nachkommastellen an.

## 3.7 Übung

## 3.7.1 Struktogramm

<b>Ausgabe ('3', 'einzelne', 'Wörter')</b>
<b>Ausgabe (1, 2+            3, 4 - 5 * 6)</b>
<b>Ausgabe ('a', 'A', '@', '□', ' ', '""', 'Ulf''s')</b>
<b>Ausgabe (1, 1.0, 5 / 9)</b>
<b>Ausgabe (2=2, 4&lt;3, false)</b>

- Was macht dieses Programm?
- Wo fängt die Programmausführung an, wo hört sie auf?
- Übersetzen Sie es nach TurboPascal.
- (Auf kariertem Papier:) Welche Ausgabe wird erzeugt?
- Wie sieht ein gewolltes Leerzeichen aus?

## 3.7.2 Ausgabe

Schreiben Sie ein Struktogramm und ein Programm, die die folgende Ausgabe erzeugen:

```

alle spaltenbreiten           =           15
sic      transit      gloria   mundi.
vierwortige      saetze      sind   langweilig.
das      alphabet:   abcdefghijklmnopqrstuvwxyz
2+2      =           4.00000000000000
128*128      =           16383.00000000000000

```

## 3.7.3 Rechenaufgaben

Schreiben Sie die folgenden Rechenaufgaben in Pascal:

$$\text{a) } \frac{4}{3} + 1,5 \quad \text{b) } 4,2 a^2 - \frac{5}{2} b \quad \text{c) } \frac{2a(5+3b)}{3c+d}$$

$$\text{d) } \sqrt{7} * \frac{a^2}{4x} : 0,2 b^2$$

## 4 Variablen

### 4.1 Datentypen

Bezeichnung	Beschreibung	Wertebereich
REAL	Zahlen <i>mit</i> Nachkommastellen, positiv, negativ, 0	-2.8E39 - 1.7E39
INTEGER	ganze Zahlen (ohne Nachkommastellen)	-32768 - 32767
CHAR	ein einzelnes Computerzeichen	'@', 'A', '_'
STRING	Kette von 0..255 Computerzeichen. '1a34' + '12a4' = '1a3412a4'	"', '@A1234', 'Hartmann'
BOOLEAN	Wahr oder falsch, männlich oder weiblich, grün oder rot, an oder aus, ...	TRUE FALSE

#### 4.1.1 Definieren im Programmkopf

```
VAR
  i : INTEGER;
  r : real;
  vorname, nachname : STRING;
  c1, c2 : CHAR;
  debug : boolean;
```

#### 4.1.2 Zuweisen im Hauptprogramm

```
i := 3; (* i ist jetzt 3 *)
vorname := 'Ulf';
nachname := 'Meyer';
c1 := ' '; (* c1 entaelt ein leerzeichen *)
c2 := ','; (* c2 enthaelt ein Hochkomma *)
i := i + 1; (* i+1 wird berechnet, das *)
(* ergibt 2, und das wird i zugewiesen. *)
(* i geht davon nicht kaputt. was vorher *)
(* in i war, ist für immer verschwunden. *)
vorname := c2; (* Variablen vom Typ STRING darf man *)
(* ein CHAR zuweisen. *)
vorname := nachname + c1 + c2;
i := i * (-2); (* i*(-2) wird berechnet, das ergibt -4, *)
(* und das wird i zugewiesen. *)
r := i; (* REAL -Variablen können INTEGER Werte *)
(* aufnehmen. *)
nachname := vorname + nachname + vorname;
```

#### 4.1.3 Schreibtischttest

Damit man den Überblick behält, welcher Wert gerade wo drin ist, macht man eine Tabelle<sup>2</sup>. In den Spaltenköpfen stehen die Variablen-Namen, in den Spalten die Variablen-Werte. Beispiel:

```

i   vorname  nachname  c1
3   Ulf      Meyer     ein leerzeichen
4                                     ein hochkomma
...
```

Überschriebene Werte sollte man durchstreichen.

<sup>2</sup>wahlweise von oben nach unten oder von links nach rechts.

#### 4.1.4 Zuweisung im Struktogramm

```
i ← 3
```

Üben Sie das jetzt anhand der Beispiele.

#### 4.2 Deutsche Umlaute und Division

i	132	148	129	225	142	153	154
CHR(i)	ä	ö	ü	ß	Ä	Ö	Ü

**Das Ergebnis einer Division ist immer REAL.**

#### 4.3 Ausgabe

```
writeln (i);
writeln (i:9);
writeln ('i ist ', i, ', und Phantom heisst real', vorname, nachname);
writeln (c2, 'CHR(148)', CHR(148));
writeln (r, r:10, r:10:5);
```

##### 4.3.1 Ausgabe im Struktogramm

```
Ausgabe (i)
```

Üben Sie das jetzt anhand des Beispiels.

#### 4.4 Übung

Was wird das folgende Programm ausgeben?

```
PROGRAM ausgaben;
```

```
VAR
```

```
  i : INTEGER;
  r : REAL;
```

```
BEGIN
```

```
  WRITELN ('grm, 01.01.2001, ein paar ausgaben.');
```

```
  i := 9;
```

```
  r := 9;
```

```
  WRITELN (i);
```

```
  WRITELN (i:8);
```

```
  WRITELN (r);
```

```
  WRITELN (r:8);
```

```
  WRITELN (r:8:5);
```

```
  WRITELN (i*r:20);
```

```
  WRITELN ('Enter to continue...');
```

```
  READLN;
```

END.

#### 4.5 Einlesen von Variablen von der Tastatur

<b>Eingabe (i)</b>
--------------------

 Es ist der Rechner, der da liest.

```
READLN (i); (* schlecht. *)
```

```
WRITELN ('Gib mir einen Wert fuer meine Zaehlvariable:');
```

```
READLN (i); (* gut. *)
```

## 4.6 Übung

Bei allen Aufgaben möchte ich Struktogramme und Quelltext haben. Exponentialdarstellung ist verboten. Drei Nachkommastellen sind o.k.

### 4.6.1 Hallo

fragt nach dem Namen des Users und sagt dann freundlich 'Hallo User.'

### 4.6.2 Zwei Zahlen

fragt nach zwei Zahlen a und b und gibt dann aus: "Die zweite Zahl war ... Die erste Zahl war ... a+b ist ... a-b ... a\*b ... a/b ... a\*a ... b\*b ..."

### 4.6.3 Würfel

fragt nach der Kantenlänge eines Würfels und gibt Volumen, Oberfläche, Raumdiagonale und Gesamtkantenlänge aus.

### 4.6.4 Brutto

fragt nach Netto und Umsatzsteuersatz und errechnet das Brutto.

### 4.6.5 Quader1

wie Würfel, bloß mit einem Quader.

### 4.6.6 Prozent-Anteil

fragt nach 4 INTEGER-Zahlen und gibt deren prozentualen Anteil an der Summe aus.

Zahl 1	13	Prozent	8.13
Zahl 2	113	Prozent	70.63
Zahl 3	3	Prozent	1.88
Zahl 4	31	Prozent	19.38
-----			
Gesamt	160	Prozent	100.00

### 4.6.7 Kreis

fragt nach der bekannten Größe (Radius, Umfang oder Fläche) und gibt die beiden unbekanntten Größen aus.

## 5 Funktionen - eingebaut

Eine Funktion ist ein Unterprogramm, das

- 0 bis viele Parameter (auch: Argumente) übernimmt und
- 1 (einen) Wert zurückgibt.

### 5.1 Unäre Funktionen (nur ein Argument)

Name	Beschreibung	Datentypen	Beispiele
CHR	wandelt ein INTEGER 0..255 in das zugehörige CHAR. Von 0 bis 31 gibt das nur Steuerzeichen.	BYTE rein, CHAR raus.	CHR(132) gibt ä
ORD	wandelt ein CHAR in das zugehörige INTEGER.	CHAR rein, BYTE raus.	ORD('ä') gibt 132
ABS	wirft das Vorzeichen weg.	Zahl rein, Zahl raus.	abs(-17) = abs(17)
TRUNC	schneidet Nachkommastellen ab.	REAL rein, INTEGER raus.	TRUNC(1.5) = 1 TRUNC(-1.5) = -1
SIN / COS / ATN	bildet den Sinus / Cosinus / ArcusTangens. Der Winkel wird im Bogenmaß gemessen: 0=0, 90 Grad = $\pi/2$ , 180 Grad = $\pi$ , 360 Grad = $2\pi$ , usw.	INTEGER oder REAL rein, REAL raus.	tan wird gebildet aus SIN/COS.
SQRT	zieht die Wurzel, oder steigt mit Fehler aus.	REAL oder INTEGER rein, REAL raus.	SQRT(-13)
LN	natürlicher Logarithmus	REAL 2 REAL	LN(2.7182818)
EXP	berechnet $e^x$ .	REAL 2 REAL	EXP(1)

### 5.2 Binäre Funktionen (Zwei Argumente)

Division wie in der Grundschule:  $15 / 4 = 3$  Rest 3.  $16 / 4 = 4$  Rest 0.  $17 / 4 = 4$  Rest 1.

Name	Beschreibung	Datentypen	Beispiele
DIV	bildet die ganzzahlige Division. Nachkommastellen werden weggeworfen.	alles INTEGER	3 DIV 4 = 0 4 DIV 4 = 1 5 DIV 4 = 1
MOD	bildet den Rest der ganzzahligen Division.	alles INTEGER	3 MOD 4 = 3 4 MOD 4 = 0 5 MOD 4 = 1

### 5.3 Die Konstante PI

PI ist in TurboPascal eingebaut. Sie wird verwendet wie eine Konstante.

## 5.4 Übung

Schreiben Sie ein Struktogramm und anschließend ein Programm, das

### 5.4.1 Beispiele

- "Schön prüft häßlich" ausgibt;
- nach einem Buchstaben fragt und seinen ascii-Code ausgibt;
- a auf 6 setzt und  $\sin(a)$ ,  $\cos(a)$ ,  $\text{atn}(a)$  und  $\text{wurzel}(a)$  ausgibt;
- $\exp(0)$  und  $\exp(1)$  berechnet:  
"exp(0) = ..... ; exp(1) = ....."
- Zwei Zahlen a und b eingeben läßt und ausgibt: a, b, a mod b, a div b,  $b \cdot (a \text{ div } b) + (a \text{ mod } b)$

### 5.4.2 wuerfel2

zu gegebener Kantenlänge a Oberfläche, Volumen und Länge der Raumdiagonalen ausgibt;

### 5.4.3 quader1

zu den gegebenen Kantenlängen a, b und c Oberfläche, Volumen und Länge der Raumdiagonalen ausgibt;

### 5.4.4 prozent1

einen eingegebenen Betrag um eine eingegebene Prozentzahl vermehrt;

### 5.4.5 math1

zu einer eingegebenen Zahl  $\sin$ ,  $\cos$ ,  $\tan$ ,  $\ln$  und Quadratwurzel berechnet und übersichtlich ausgibt;

### 5.4.6 Benzinverbrauch

das nach Eingabe von gefahrener Strecke (in km) und verbrauchtem Benzin (in l) den Benzinverbrauch errechnet und ausgibt.

### 5.4.7 quadrg1

das die quadratische Gleichung  $y = ax^2 + bx + c$  löst (ohne Fallunterscheidung).

## 6 Bedingte Verzweigung

Rechner wären langweilig, wenn sie nicht Fallunterscheidungen treffen könnten, wie zB

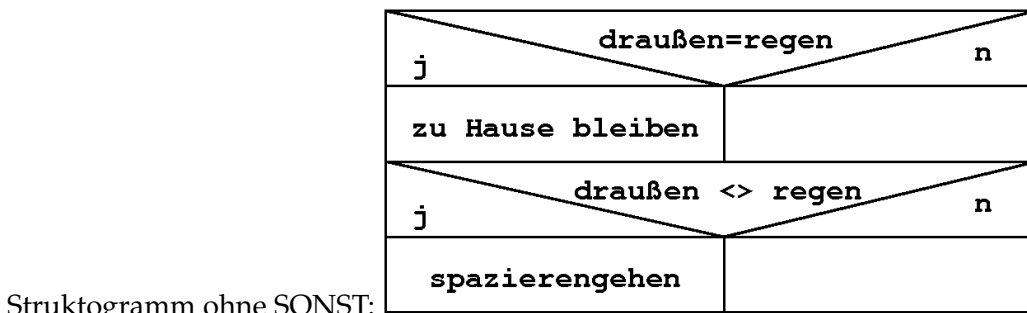
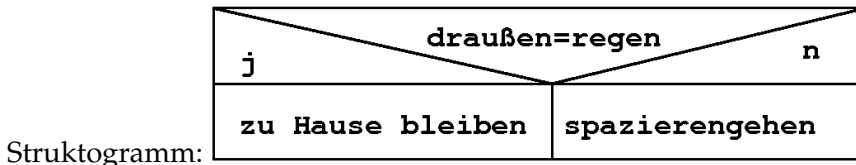
"Sie haben eine Kantenlänge < 0 eingegeben. Fehler!"

"Ich darf keine Wurzel aus einer negativen Zahl ziehen."

"Ich darf nicht durch 0 teilen. *Niemals!*"

### 6.1 Einfaches Beispiel: Regen

WENN es regnet, DANN bleibe ich zu Hause, SONST gehe ich spazieren.



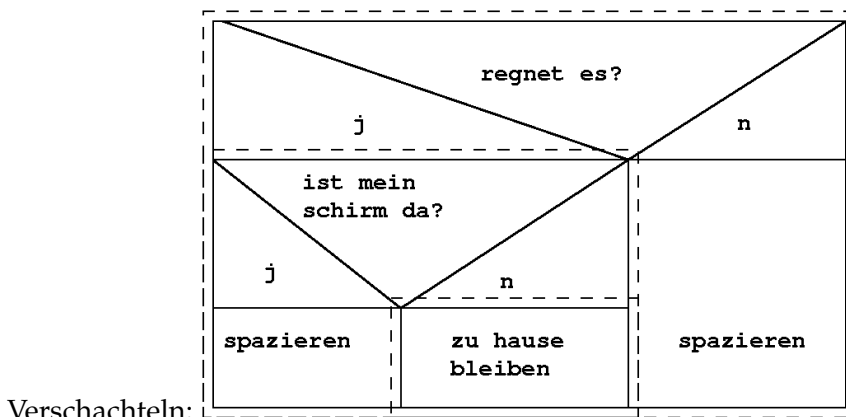
Wenn die Bedingung erfüllt ist, wird der "Ja"-Kasten ausgeführt, sonst der "Nein-Kasten.

Die Bedingte Verzweigung ist selbst ein Kasten und darf überall dort stehen, wo auch ein Kasten stehen darf.

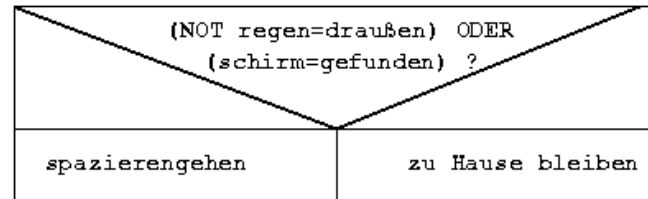
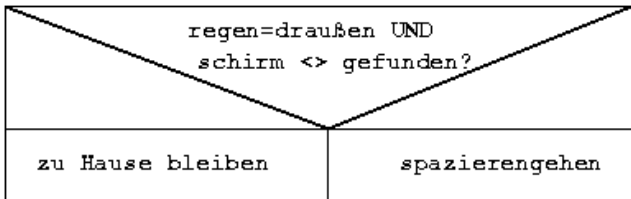
### 6.2 Verschachteltes Beispiel: Regen und Schirm

Wenn es NICHT regnet ODER ich endlich meinen Schirm finde, gehe ich spazieren, sonst bleibe ich zu Hause.

Wenn es regnet UND mein Schirm ist NICHT da, bleibe ich zu Hause, IN JEDEM ANDEREN FALL gehe ich spazieren.



Verknüpfen:



**Eine IF-THEN-ELSE-Anweisung ist ein Kasten und darf überall dort stehen, wo ein Kasten stehen darf.**

**Eine IF-THEN-ELSE-Anweisung enthält 2 Kästen. Diese dürfen alles enthalten, was ein Kasten enthalten darf. ZB Zuweisungen, Ein- und Ausgaben, oder weitere IF-THEN-ELSE-Anweisungen.**

**Übung:** Malen Sie die Kästen bunt an.

**Abhängig von der Bedingung geht das Programm entweder den linken oder den rechten Weg. Niemals beide. Niemals keinen.**

**Übung:** Denken Sie sich in verschiedenen Farben Startbedingungen aus, und maen Sie die entsprechenden Wege bunt an.

### 6.3 Bedingung

Eine Bedingung ist BOOLEAN, d.h. entweder TRUE oder FALSE.

**Beispiele**

(i=4)	(r=4)	(c1<c2)
(stri1>stri2)	(TRUE <> FALSE)	(1<2) AND (3<4)
((1<2) AND (3<4)) OR (5>6)	(1<2) AND ((3<4) OR (5>6))	(1<2) AND ((3<4)) OR (5>6)

### 6.4 Syntax

```
IF (draussen=regen)
THEN BEGIN
  zu_hause_bleiben;
  kaffe_trinken;
END [ ELSE BEGIN
  schuhe_und_jacke_an;
  spazierengehen;
END ] ;
```

**Vor ELSE steht kein Semikolon.**

## 6.5 AND, OR und NOT

**AND, OR und NOT haben eine höhere Priorität als  $<$ ,  $>$ ,  $<>$ ,  $=>$  etc.  
Damit der Compiler alles richtig erkennt, müssen wir alle Gleichheits- und Ungleichheits-Bedingungen einklammern.**

**Übung:** Füllen Sie die Tabelle aus.

a	b	(a and b)	(a or b)	(not a)	not (not (a) and not (b))
	+	+	+	+	+
true	true				
true	false				
false	true				
false	false				

6.6 Musterlösungen der ersten drei Übungen

Eingabe (Zahl)	
Zahl < 0 ?	
J	N
Ausgabe (zahl, ' ist kleiner als Null!')	
Zahl = 0 ?	
J	N
Ausgabe (zahl, ' ist gleich Null!')	
Zahl > 0 ?	
J	N
Ausgabe (zahl, ' ist größer als Null!')	

Eingabe (Zahl)		
Zahl < 0 ?		
J	N	
Ausgabe (zahl, ' ist kleiner als Null!')	Zahl = 0 ?	
	J	N
Ausgabe (zahl, ' ist gleich Null!')	Ausgabe (zahl, ' ist gleich Null!')	Ausgabe (zahl, ' ist größer als Null!')

geschachtelt

ungeschachtelt

Eingabe (Gehalt)	
3,5 Erhöhung <-- --- * Gehalt 100	
Erhöhung < 200 ?	
J	N
Erhöhung <-- 200	
Ausgabe ('Neues Gehalt: ', Gehalt + Erhöhung)	

Eingabe (Name1, Name2)	
Name1 < Name2 ?	
J	N
Ausgabe (Name1, ' kommt vor ', Name2)	Ausgabe (Name2, ' kommt vor ', Name1)

## 6.7 Übung

Zeichnen Sie stets zunächst ein Struktogramm. Übersetzen Sie anschließend das Struktogramm nach Pascal.

### 6.7.1 Zahlenterror

Schreiben Sie ein ungeschachteltes und ein geschachteltes Programm, das untersucht, ob eine eingegebene REAL-Zahl  $<0$ ,  $=0$  oder  $>0$  ist.

### 6.7.2 Gehalt

Die Mitarbeiter der BIC KG bekommen eine Gehaltserhöhung: +3,5%, aber mindestens +50 EUR. Das Programm soll nach Eingabe des Gehalts die Gehaltserhöhung und das neue Gehalt ausgeben.

### 6.7.3 Namen

Schreiben Sie ein Programm, das von 2 eingegeben Namen die richtige Ausgabe "Name1 kommt vor Name2." erzeugt.

### 6.7.4 Bauspar

Beim Bausparen erhalten Verheiratete für Sparleistungen bis zu 1600DM eine Prämie, Alleinstehende nur bis zur Hälfte dieses Betrages. Die Prämie beträgt 14% der Sparleistung. Für jedes Kind werden 2% zusätzlich gewährt. Es gibt halbe Kinder, aber keine negativen Kinder.

Man schreibe ein Programm, das nach Eingabe von Sparleistung, Familienstand und Kinderanzahl die Prämie ermittelt.

### 6.7.5 Bank

Eine Bank führt bei jedem Girokonto die ersten 10 Buchungen kostenlos aus. Für die nächsten 10 berechnet sie 30 Cent pro Buchung, für jede weitere 20 Cent.

Schreiben Sie ein Programm, das zur Anzahl Buchungen die Gebühr berechnet.

### 6.7.6 QuadrG12

Es werde die allgemeine quadratische Gleichung  $x^2 + px + q = 0$  gelöst.

Berücksichtigen Sie die Sonderfälle  $p=0$  und/oder  $q=0$ .

### 6.7.7 DreiName

Drei eingegebene Namen sollen in alphabetischer Reihenfolge ausgegeben werden.

### 6.7.8 Geradengleichung

fragt nach  $m$  und  $b$  und gibt die Nullstelle aus und ob der Graph steigend oder fallend ist.

## 7 Wiederholung 1

1. Schreibe hinter die Aussagen, die deiner Ansicht nach falsch sind, ein f. Hinter die richtigen mach bitte einen Haken.
2. In der gesamten Klasse wird diskutiert, welche richtig oder falsch sind. STREICHE DIE FALSCHEN AUSSAGEN ODER AUSSAGENTEILE DURCH.
3. Wiederhole zu Hause (in der U-Bahn, im Wartezimmer,...) das, was du falsch angekreuzt hattest.

- 1 Pascal-Programme sind zum Lösen von Problemen da.
- 2 Mit Pascal lassen sich alle Probleme der Welt lösen.
- 3 Wer die Problemstellung der Aufgabe nicht verstanden hat, kann auch kein passendes Pascal-Programm dazu schreiben, sondern muß sich das genauer erklären lassen.
- 4 Das Vorgehen bei der Lösung eines Problems heißt Algorithmus.
- 5 Struktogramme bilden Algorithmen in einer computernahen Form ab.
- 6 Struktogramme helfen bei der Erstellung größerer Programm-Projekte.
- 7 Ich sollte nie/immer meinen Namen in meinen Pascal-Quelltext schreiben.
- 8 Ich sollte nie/immer das aktuelle Datum in meinen Pascal-Quelltext schreiben.
- 9 Ich sollte beides in Kommentarzeichen ((\* \*) oder { }) einklammern.
- 10 Variablennamen sollten markant und "sprechend" sein.
- 11 Das kürzeste lauffähige Pascal-Programm lautet program p; begin end.
- 12 Ich muß immer clrscr benutzen.
- 13 Wenn ich clrscr benutzen will, muß ich die Unit crt mittels uses crt einbinden.

Streiche die ungültigen Werte durch.

- 14 Gültige Werte für integer-Variablen sind: 1 -32000 32000.0
- 15 Gültige Werte für real-Variablen sind: sin(3.142) 3/7 0,2
- 16 Gültige Werte für string-Variablen sind: ' ' 'alles klar!!!'  
'HM' 'Alles @\$\$'!)/&%&\$\$!' 'Das Ergebnis ist : '
- 17 Gültige Werte für boolean-Variablen sind: TRUE (3>4) OR (' ' < 'h')  
0 FALSE -1 '-1'.
- 18 Gültige Werte für char-Variablen sind: A 'A' B 'B' 'äöü?'

readln (a,b) (a und b vom Typ real) macht folgendes:

- 19 Wenn ich eintippe 17.0 20.4, sagt es division by zero error.
- 20 Wenn ich eintippe 17.0 20.4, weist es a 17 zu, b 20,4 zu, und macht weiter im Programm.
- 21 Wenn ich eintippe 17.0<Enter-Taste>20.4, sagt es division by zero error.
- 22 Wenn ich eintippe 17.0<Enter-Taste>20.4, weist es a 17,4 zu, b 20 zu, und macht weiter im Programm.

- 23 In einer Zuweisungsanordnung im Struktogramm steht links eine Formel, dann ein Pfeil nach rechts, und dann ein Variablenname.
- 24 In einer Zuweisungsanordnung im Struktogramm steht bloß eine Formel.
- 25 In einer Zuweisungsanordnung im Struktogramm steht links ein Variablenname, dann ein Pfeil nach links, und dann eine Formel, deren Datentyp auf keinen Fall derselbe wie bei der Variable ist.

- 26 sin(x) berechnet mir den Sinus von x. x ist dabei im Bogenmaß, d.h. 2\*PI sind ein Vollkreis.
- 27 cos(x) berechnet mir den Cosinus von x. x ist dabei vom Typ string, das Ergebnis der Berechnung ist vom Typ boolean.
- 28 tan(x) berechnet mir den Tangens von x. x ist dabei im Bogenmaß.

```
29 sqrt(x) berechnet mir das Quadrat von x. Ergebnis ist vom Typ integer.
30 13 div 5 berechnet mir 13/5 und schneidet die Nachkommastellen ab.
   Ergebnistyp = integer.
31 13 mod 5 berechnet den Rest der ganzzahligen Division von 13/5, also 3.

32 writeln (a) (a vom Typ real) liefert ein korrektes, gut lesbares Ergebnis.
33 writeln (3 > 4:80) schreibt FALSE rechts an den Rand des Bildschirms.
34 writeln ('3 > 4':80) schreibt FALSE an den rechten Rand des Bildschirms.
35 writeln ('3 > 4':80) compiliert nicht.
36 writeln (3>4:80) schreibt 3 > 4 rechts an den Rand des Bildschirms.
37 writeln ('Irgend', 1.0:1:0, ' ', 'Blöd'+sinn') schreibt irgendeinen
   Blödsinn.
38 i:=1; writeln ('Irgend', 1.0:i:i, ':1,'Blöd'+sinn') (i vom Typ
   integer) compiliert nicht.
39 if a>b then begin writeln ('a>b!') end else begin writeln ('a<=b!') end;
   kann ich genausogut lesen und auf Fehlerfreiheit testen wie
if (a > b) then begin
   writeln ('a>b!')
end else begin
   writeln ('a<=b!')
end;
```

## 8 Zählschleife

Falls der Rechner etwas mehrfach machen soll, ist es mühsam, den Quelltext auszuschneiden und 5mal einzufügen. Und unmöglich, ihn 5000mal einzufügen.

Besser wäre es, wir hätten folgende Struktur:

“Laß die Variable `index` von `startwert` bis `endwert` mit Schrittweite 1 laufen.

Mach dabei jedesmal, wenn du `index` erhöhst hast, irgendwas.”

```
setze schleifenzaehler auf startwert
SCHLEIFE
  mach irgendwas
  erhoehe den Schleifenzaehler um 1
  WENN schleifenzaehler=endwert
    DANN gehe nach SCHLEIFENENDE
    SONST gehe nach SCHLEIFE
SCHLEIFENENDE
```

<b>FÜR i := startwert TO endwert TUE</b>
--

<b>was auch immer...</b>
--------------------------

Und das gibt es. Struktogramm:

### 8.1 Syntax

```
FOR i := 1 TO 9999 DO BEGIN
  tue_irgendwas;
END;
```

**Der Zähler ist eine Variable vom Typ INTEGER.**

`startwert` und `endwert` sind Ausdrücke vom Typ INTEGER.

**Schrittweite ist immer 1.**

**Oder -1, wenn man nicht TO, sondern DOWNTO verwendet.**

## 8.2 Übung

Zu jedem Programm gehört ein Struktogramm.

### 8.2.1 Zahlen zusammenzählen

Was ist die Summe aller Zahlen von 1 bis 100?

### 8.2.2 FORDO01

Das folgende Programm soll 20 Zeilen ausgeben:

in der ersten Spalte die Zahlen 1..20, in der zweiten  $1^2$  bis  $20^2$ ,

in der dritten  $1^3$  bis  $20^3$ , in der vierten  $1^4$  bis  $20^4$ .

Es sind integer zu verwenden. Spaltenbreite = 10.

Kommentieren Sie das Ergebnis schriftlich, und geben Sie den Kommentar in einer weiteren Programmzeile aus.

### 8.2.3 FORDO02

Schreiben Sie ein Struktogramm und anschließend ein Programm, das 21 Zeilen ausgibt:

in der ersten Spalte die Zahlen -5 bis 5 in Schrittweite 0,5,

in der zweiten Spalte deren Quadrat,

in der dritten Spalte TRUNC des Wertes der 1. Spalte,

in der 4. Spalte den Kehrwert der ersten Spalte (sofern möglich).

Spaltenbreite ist 8 bei optimaler Lesbarkeit des Pascal-Quelltextes und der Ausgabe.

### 8.2.4 Fakultät

Schreiben Sie ein Struktogramm und anschließend ein Programm, das  $n!$  (lies: enn Fakultät) berechnet. Das Programm soll  $10!$  fehlerfrei berechnen.  $n$  wird eingegeben.

Hinweis:  $0! = 1$ .  $n! = 1 * 2 * 3 * .. * (n-1) * n$  für  $n > 0$ .

### 8.2.5 Großes Einmaleins

Schreiben Sie ein Struktogramm und anschließend ein Programm, das das große Einmaleins (von  $1*1$  bis  $20*20$ ) in Tabellenform auf den Bildschirm schreibt.

### 8.2.6 ASCII-Tabelle

Schreiben Sie ein Programm mit der Laufvariable `lauf`, das die Zahlen von 32 bis 255, jeweils gefolgt

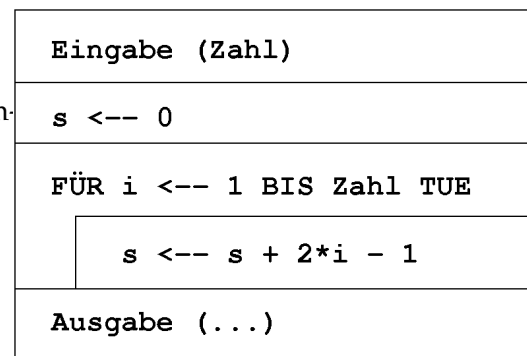
von einem Leerzeichen, `CHR(lauf)` und noch drei Leerzeichen, in übersichtlichen Spalten auf den Bildschirm schreibt.

### 8.2.7 SINUS-Verlauf

Schreiben Sie ein Programm, das einen um  $90^\circ$  gedrehten Sinus-Verlauf von 0 bis  $2*PI$  auf den Bildschirm malt. Benutzen Sie dazu die Formatierung innerhalb einer `writeln`-Anweisung.

### 8.2.8 WasIstDas

Gegeben ist ein Struktogramm:



- Schreibtischtest für  $Zahl = 1,3,4$ .
- Ersetzen Sie (...) durch etwas Sinnvolles.
- Schreiben Sie das Pascal-Programm dazu.

### 8.2.9 Einfacher

```

... (* Kopf verlorengegangen *)
summe := 0;
for i:=1 to n do begin
  summe := summe + 2*i + 1;
  writeln (n:10, summe:10);
end;
readln;
end.

```

- Ergänzen Sie den fehlenden Programmkopf.
- Erzeugen Sie eine Bildschirmausgabe für  $n=5$ .
- Schreiben Sie eine einfachere Version.

## 9 Unit CRT

### 9.1 Units

Units ("Bausteine") können die Funktionalität von TurboPascal erweitern.

Man muß sie einbinden mittels USES, gleich hinter dem Programm-Namen:

```
PROGRAM vielrot;
USES crt;                (* Units werden mit USES eingebunden *)

BEGIN
  SetTextBackground (red);
  clrscr;
END.
```

### 9.2 Probleme mit mehr als 100 MHz

Auf Rechnern > 100 MHz funktioniert Unit crt NICHT.

Im Web gibt es aber Patches.

### 9.3 More Information

Wer Genaueres wissen möchte, chreibt zB `textcolor` irgendwo insein Programm, setzt den Cursor unter einen der Buchstaben, und drückt Ctrl-F1.

### 9.4 CRT-Aufrufe, die wir nie benutzen

Name	Parameter	???	Beschreibung
AssignCrt	Name des CRT-Fensters und Name der Textdatei	–	Ordnet dem CRT-Fenster eine Textdatei zu.
ClrEol	n.v.	–	Löscht alle Zeichen von der momentanen Cursor-Position bis zum Zeilenende.
DelLine	n.v.	–	Löscht die Zeile auf der der Cursor positioniert ist.
InsLine	n.v.	–	Fügt eine Leerzeile an der Position des Cursors ein.
TextMode	n.v.	–	Legt einen bestimmten Textmodus fest.
WhereX	n.v.	Funktion	Liefert die X-Koordinate (Zeilenposition) des Cursors zurück.
WhereY	n.v.	Funktion	Liefert die Y-Koordinate (Spaltenposition) des Cursors zurück.
Window	n.v.	–	Definiert einen Bereich des Bildschirms als Textfenster.

## 9.5 CRT-Aufrufe, die wir selten benutzen

Name	Parameter	Rückgabe-Datentyp	Beschreibung
ClrScr	keine	–	Löscht den Bildschirm und setzt den Cursor in die linke obere Ecke.
Delay	Anzahl der Millisekunden(integer)	–	Wartet die angegebene Anzahl von Millisekunden.
GotoXY	X-Koordinate (1..80), y-Koordinate (1..25), beide integer	–	Positioniert den Cursor auf gegebene Koordinaten innerhalb eines virtuellen Bildschirms.
HighVideo	keine	–	Setzt "hohe Intensität" für die Zeichenausgabe.
KeyPressed	keine	boolean	Prüft, ob eine Taste gedrückt wurde.
LowVideo	keine	–	Setzt "niedrige Intensität" für die folgenden Zeichenausgaben.
NormVideo	keine	–	Setzt das Textattribut für nachfolgende Ausgaben, das beim Start des Programms gesetzt war.
NoSound	keine	–	Schaltet den eingebauten Lautsprecher ab.
ReadKey	keine	char	Liest ein Zeichen von der Tastatur.
Sound	Frequenz des Tones in Hz	–	Aktiviert den eingebauten Lautsprecher.
TextBackground	0..7 oder white, yellow, cyan, magenta, red, blue, green, black	–	Legt die Hintergrundfarbe für folgende Textausgaben fest.
TextColor	dito	–	Legt die Zeichenfarbe für folgende Textausgaben fest.

## 9.6 Übung CRT

### 9.6.1 Bunt

Schreiben Sie ein Programm, das

- den Bildschirm löscht,
- Dies ist roter Text auf gelbem Hintergrund ausgibt,
- Dies ist grüner Text auf cyan Hintergrund ausgibt,
- eine halbe Sekunde mit 100 Hz piept,
- Drücke irgendeine Taste mitten auf dem Bildschirm ausgibt,
- der char-Variablen `c1` den Wert von `readkey` zuweist,
- ausgibt Sie haben die Taste ... gedrückt,
- und sich dann beendet.

### 9.6.2 Würfel3

Verändern Sie das Programm `wuerfel2.pas` wie folgt: Zunächst fragt es nach Wunsch-Textfarbe und Wunsch-Hintergrundfarbe, anschließend setzt es sie, dann erst fängt es an zu rechnen.

### 9.6.3 Quader3

Machen Sie Ihre Quaderberechnung bunt, und addieren Sie Getöse.

## 10 Schleifen mit Start- oder Stopbedingung

Die FOR-DO-Schleife ist sehr praktisch, wenn man weiß, wie viele Durchläufe zu erledigen sind.

Bei Näherungsrechnungen oder bei menügesteuerten Programmen kennt man die Anzahl der Schleifendurchläufe jedoch nicht im voraus.

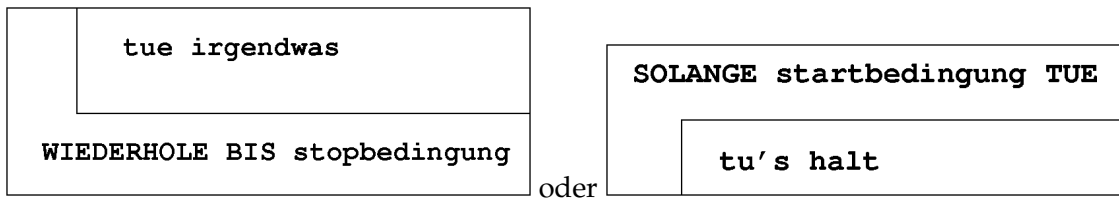
Wie schön wäre es, wenn man sagen könnte:

```
FANG_AN
  mach irgendwas      (* sollte Einfluss auf die Stopbedingung haben *)
WENN NICHT stopbedingung DANN gehe_zu FANG_AN
```

oder

```
SOLANGE startbedingung TUE BEGIN
  machs halt          (* sollte Einfluss auf die Startbedingung haben *)
END
```

### 10.1 Struktogramm



### 10.2 Syntax

```
REPEAT
  mach irgendwas      (* sollte Einfluss auf die Stopbedingung haben *)
UNTIL stopbedingung;
```

oder

```
WHILE startbedingung DO BEGIN
  mach irgendwas      (* sollte Einfluss auf die Startbedingung haben *)
END;
```

### 10.3 Übung

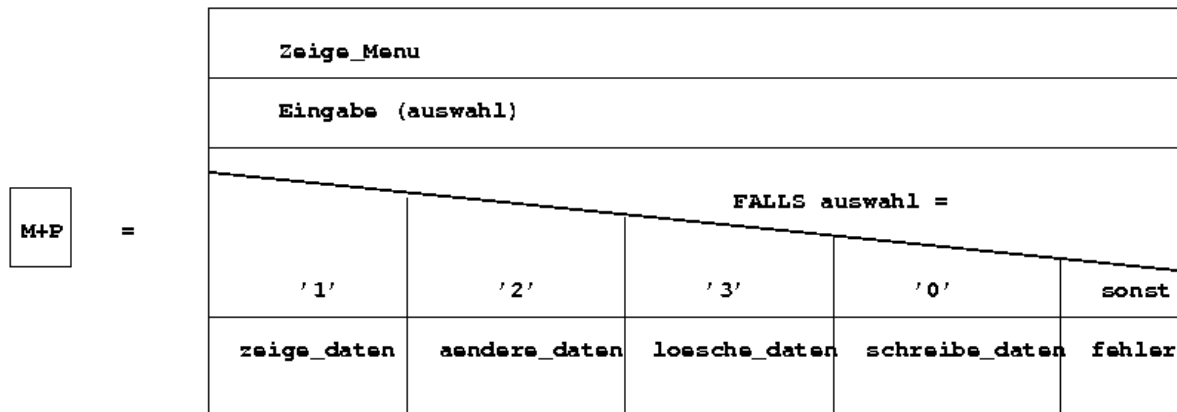
Zu jedem Programm gehört ein Struktogramm.

#### 10.3.1 Summe der $1/2^n$

Wieviel ist die Summe von  $1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \dots$  ? Berechnen Sie auf  $10^{-6}$  genau.

#### 10.3.2 Menüführung

Das Struktogramm für Menu+Arbeiten gebe ich vor:



Entwerfen Sie das zugehörige Struktogramm a) mit REPEAT-UNTIL. b) mit WHILE-DO.

#### 10.3.3 Simulation

Simulieren Sie eine FOR-DO-schleife inclusive Ausgabe der Laufvariable mittels einer

a) REPEAT-UNTIL-Schleife; b) WHILE-DO-Schleife.

#### 10.3.4 Barbie's Primzahltest

**Eine Primzahl ist eine ganze Zahl, die sich ohne Rest nur durch 1 und sich selber teilen läßt.**

**Ein LONGINT ist ein 32-bit-INTEGER mit einem Wertebereich von  $-2^{31}$  bis  $2^{31}-1$  entsprechend -2147483648 bis 2147483647.**

```
Willkommen zu Barbie's Primzahltest.
Geben Sie Ihre Zahl ein: 2387412893469
Teiler: 23478
Teiler: 123478956
2387412893469 ist keine Primzahl.
Nochmal [j] ? N
Tschuess.
```

#### 10.3.5 Ken's Quader

Bevor Sie das Struktogramm schreiben, überlegen Sie, ob der User Punkt 3 als erstes aufrufen darf.

Ken's Benutzerfreundliche Quaderberechnung

1 Oberfläche anzeigen

- 2 Volumen anzeigen
- 3 Kantenlänge anzeigen
- 4 Kantenlängen neu festlegen
- 0 Exit

Bitte wählen Sie:

### 10.3.6 Summe aller $1/x^2$

Schreiben Sie ein Programm, das die Summe aller  $1/x^2$  für  $x$  von 1 bis ??? berechnet und diese Summe anzeigt, sobald  $1/x^2$  kleiner als 0,000001 ist.

Falls Sie  $1/x^2$  innerhalb des Programms berechnen müssen, schreiben Sie den erhaltenen Wert in die Variable gonzo und benutzen diese weiterhin.

### 10.3.7 Iteratives Wurzelziehen nach Hans

Wenn ich die Quadratwurzel aus  $Z$  ziehen möchte, funktioniert folgendes Verfahren:

**Ich fange an mit  $W = Z/2$ .**

**Ich berechne ein  $\Delta = \frac{Z-W^2}{2*W}$  und ein verbessertes  $W$  aus  $W + \Delta$ .**

**Das mache ich, bis  $\Delta$  kleiner als die benötigte Genauigkeit ist.**

Der Radikant wird vom User eingegeben.

Während der Berechnung wird  $i$ ,  $w$ ,  $w*w$  und  $\Delta$  fortlaufend zeilenweise ausgegeben.

Am Schluß wird Anzahl der Iterationsschritte, Radikant, Wurzel,  $W^2$ ,  $\Delta$  ausgegeben.

Benutzen Sie den Datentyp EXTENDED. EXTENDED ist ein Datentyp mit Nachkommastellen. Geben Sie 18 Nachkommastellen aus.

### 10.3.8 Iteratives Wurzelziehen nach Lisa

**Ich fange an mit  $a=Z$  und  $b=1$ . So ist  $a*b = Z$ .**

**Ich berechne ein verbessertes  $a$  aus  $(a+b) / 2$  und ein neues  $b$  aus  $Z / a$ .**

**Das mache ich, bis  $(a$  genügend nahe an  $b)$ .**

### 10.3.9 Diophant

521 Personen sollen befördert werden. Es stehen Busse mit 21, 41 oder 43 Sitzplätzen zur Verfügung.

Finden Sie heraus, wieviele Busse welchen Typs optimalerweise gebraucht werden.

## 11 Wrapup Schleifen und Random [optional]

### 11.1 Wrapup Schleifen

Legen Sie eine Tabelle an, und vergleichen Sie die 3 bekannten Sorten Schleifen anhand folgender Fragen:

- Anzahl der Durchläufe?
- Min. Anzahl der Durchläufe?
- Max. Anzahl der Durchläufe?
- Schleifenvariable?
- BEGIN .. END?
- Abbruchbedingung?
- Startbedingung?

### 11.2 Random

Zufallszahlen sind etwas Feines:

- Man spart sich das Eintippen von Werten.
- Man kann größere Datenmengen zum Testen von Programmen erzeugen

```

RANDOMIZE;                (* mischt die Zufallszahlen *)
writeln (random);        (* gibt eine REAL Zufallszahl z.
                          0 <= z < 1 *)
writeln (random(93));    (* gibt eine INTEGER Zufallszahl
                          zwischen 0 und 92 inclusive. *)

```

#### 11.2.1 Übung Würfel1

Lassen Sie 1000mal würfeln, und bestimmen Sie die durchschnittliche Augenzahl aller Würfe.

#### 11.2.2 Übung Würfel2

Lassen Sie 3 Würfel 1000mal würfeln, und bestimmen Sie Anzahl und Prozentsatz

- a) der Würfe 6-6-6.
- b) der Würfe, wo alle Augenzahlen gleich sind;
- c) der Würfe, wo die Summe der Augenzahlen 4 ist.

## 12 Bedingte mehrfache Verzweigung

Für Wahlmöglichkeiten in Menüs gibt es noch eine andere Verzweigung.

Natürlich könnten wir immer schreiben:

```
IF eingabe = '1' THEN zeige_daten;
IF eingabe = '2' THEN aendere_daten;
IF eingabe = '3' THEN loesche_daten;
IF eingabe = '0' THEN begin
  schreibe_daten;
  quit;
end;
```

aber das Struktogramm dafür braucht viel Platz, sieht nicht besonders elegant aus, und ein ELSE-Zweig ist nur mit Mühe zu erreichen. Wie wärs hiermit:

FALLS eingabe =				
'1'	'2'	'3'	'0'	sonst
zeige_daten	aendere_daten	loesche_daten	schreibe_daten quit	fehler

### 12.1 Syntax

```
CASE eingabe OF                                (* eingabe ist eine Variable *)
  '1' : zeige_daten;
  '2' : BEGIN                                  (* '1', '2' usw. sind Konstanten *)
    aendere_daten;
  END;
  '3' : loesche_daten;
  '0' : BEGIN
    schreibe_daten;
    quit;
  END                                          (* kein ; vor ELSE *)
ELSE BEGIN
  WRITELN ('Falsche Eingabe, versuch''s nochmal.');
```

END; (\* dieses END gehoert zum ELSE \*)

END; (\* dieses END gehoert zum CASE \*)

**Jedes CASE muß durch ein END abgeschlossen werden.**

**Die überprüfte Variable muß vom aufzählenden Typ sein (INTEGER, CHAR, BOOLEAN).**

**Die Überprüfungen müssen Konstanten oder Konstantenbereiche sein:**

**3            '3'            TRUE            2..3            1,3..5,7            'a..'z','A..'Z'**

## 12.2 Übungen

Zu jeder Aufgabe gehört ein Struktogramm!

### 12.2.1 CASE01

soll untersuchen und ausgeben, ob eine eingegebene Integer-Zahl im Bereich -3 bis 1, 2 bis 2, oder irgendwo anders ist.

### 12.2.2 CASE02

untersucht und gibt aus, ob ein eingegebenes Zeichen ein Kleinbuchstabe, ein Großbuchstabe, eine Ziffer oder etwas völlig anderes ist.

### 12.2.3 Taschenrechner

Der User gibt eine Zahl, einen Operanden (+, -, \*, /) und noch eine Zahl ein, woraufhin die Eingabe sowie das Ergebnis der Berechnung ausgegeben wird.

### 12.2.4 Wissenschaftlicher Taschenrechner

Schreiben Sie ein Struktogramm und anschließend ein Programm, bei dem der User eine Zahl eingibt und dann ein Menü gezeigt bekommt, in dem er selektieren kann:

```
1 gibt den Sinus der Zahl
2 den Cosinus
3 den Tangens
4 das Quadrat
5 die Wurzel (*mit Fehlerabfangen*)
0 beendet das Programm sofort.
```

### 12.2.5 Buchung2

Schreiben Sie ein Struktogramm und anschließend ein Programm zur Berechnung von Buchungsgebühren. Die Anzahl der Buchungen wird vom User eingegeben.

Die ersten 10 Buchungen kosten nichts.

Die zweiten 10 Buchungen kosten je 30 Cent.

Alle folgenden kosten 20 Cent.

## 13 Wiederholung 2

Korrigieren Sie die falschen Aussagen. Markieren Sie die richtigen als richtig.

- o Mit der case-Anweisung lassen sich sehr schön benutzerfreundliche Menüs gestalten.
- o In das Struktogramm zu case schreiben wir am besten nur Kurznamen, und zeichnen das Struktogramm mit der Kurzbezeichnung neben oder unter das Hauptstruktogramm.

Syntax der case-Anweisung

- o Ich muß case-Anweisungen bei den Variablen definieren.
- o Das erste Wort lautet IF.
- o Dann kommt eine Bedingung.
- o Dann kommt das Schlüsselwort OFL.
- o Dahinter kommen verschiedene Zeilen. Jede fängt mit einer Formel an, dann kommt ein Gedankenstrich, dann 0..viele Anweisungen, jeweils durch ein Semikolon abgeschlossen.
- o Evtl. kommt noch ein else-Zweig, dessen Anweisungen mit begin..end geklammert werden müssen.
- o Pascal weiß selber, wann die case-Anweisung zu Ende ist. Sie braucht daher kein end.

Syntax der REPEAT-Anweisung

- o Zunächst kommt ein REPEAT, dann eine Bedingung, dann ein DO.
- o Dann kommen beliebig viele Anweisungen, die nicht mit einem Semikolon abgeschlossen werden dürfen.
- o Schließlich kommt ein UNTIL;

Syntax der WHILE-Anweisung

- o Zunächst kommt ein WHILE, dann eine Bedingung, dann ein OF.
- o Dann kommen beliebig viele Anweisungen, die nicht mit einem Semikolon abgeschlossen werden dürfen.
- o Schließlich kommt ein END.

Zufallszahlen

- o Der Zufallszahlen-Generator wird mit Randomize neu gestartet.
- o random(7) liefert eine Zufallszahl zwischen 1 und 6.
- o Die Anweisungen
 

```
a:=random(5); b:=random(5);
und
a:=random(5); b:=a;
```

 liefern dasselbe Resultat.

Felder

Wir sprechen im folgenden von einem Feld von 20 Werten, Typ real, Name des Feldes: mine.

- o Ich muß das Feld bei den Konstanten definieren.
- o Die Definition lautet:
 

```
const mine : array [1...20] of boolean
```
- o Ich darf im Hauptprogramm auf die Feldelemente zugreifen, aber nicht in den Prozeduren.
- o Die Feldelemente des Feldes heißen: mine(0), mine(1), mine(2), mine(3), mine[4], ..., mine(19).

Korrigieren Sie die Fehler, und beschreiben Sie in eigenen Worten, was jedes der Programmstücke 1 bis 5 tut.

```
1 for i=1 to 20 do mine[i] := 0;
2 for j=1 to 20 do begin i:=j; mine[i] := 0 end;
3 mine(1) := 1;
  for i:=2 to 20 do
    mine[i] := mine[i-1]*i;
4 for i:=1 to n do write (mine[i]:0:9); writeln;
5 for i:=1 to n do
  mine[i]:=2*i+1;
```

## 14 Funktionen und Prozeduren - sauber

### 14.1 Funktionen - selbstgeschrieben

Die Funktion SIN ist fantastisch. Was immer man ihr für einen Wert gibt, sie berechnet den Sinus dazu. Lästigerweise will sie den Winkel in Radianten haben.

Wie nett wäre es, wenn wir unsre eigenen Funktionen schreiben könnten, zB

```
function mysin (???)      (* gibt den Sinus des Winkels in Grad*)
function mywurzel (???)  (* berechnet die Quadratwurzel iterativ *)
```

Die gute Nachricht ist: Wir können.

Die schlechte Nachricht ist: Wir müssen vorher über das Wesen von Funktionen nachdenken.

**Eine Funktion ist ein Ding, das 0..viele Parameter aufnimmt und *einen* Wert zurückgibt.**

Parameter dürfen u.a. sein:

- keiner (zB random);
- einer (zB sin);
- zwei (zB vektor\_laenge);
- drei (zB quader\_raumdiagonale);
- viele gemischte. ZB string s, bool b für die Funktion  
zaehl\_zeichen\_mit\_oder\_ohne\_leerzeichen.

#### 14.1.1 Syntax

**Funktionen müssen im Programmkopf definiert werden, d.h. zwischen globalen Variablen und Hauptprogramm.**

```
FUNCTION summe (i, j : REAL) : REAL;
VAR
  s : REAL;
BEGIN
  s := i + j;      (* weist s die Summe der Parameter zu *)
  summe := s;     (* beendet die Funktion und gibt Wert zurück *)
END;
```

Schauen wir uns das in Ruhe an:

```
FUNCTION summe (i, j : REAL) : REAL;
```

“Hallo TurboPascal! Ich definiere eine Funktion namens summe.

Sie will immer zwei REAL, und das immer in dieser Reihenfolge. Den ersten Parameter nenne ich intern i und den zweiten j.

Heraus kommt immer etwas vom Typ REAL.”

```
VAR
  s : REAL;
```

”Ich werde außerdem eine REAL-Variable namens `s` benutzen.

- Das aufrufende Programm kann sie nicht sehen.
- Falls es eine gleichnamige Variable im aufrufenden Programm gibt, kann die Funktion diese ebenfalls nicht sehen.
- Sobald `foo` fertig gerechnet hat, wirst du diese Variable *vernichten*.”

```
s := i + j;          (* weist s die Summe der Parameter zu *)
```

Hier darf alles Mögliche stehen: Ausgaben, Eingaben, Schleifen, Verzweigungen...

```
summe := s;         (* beendet die Funktion und gibt Wert zurück *)
```

”He Pascal! Wenn du an diese Stelle kommst,

- brichst du die Funktion `summe` ab,
- tötet alle lokalen Variablen;
- und gibst dem aufrufenden Programm dies hier als Ergebnis.”

## 14.2 Prozeduren

Prozeduren werden im Hauptprogramm aufgerufen wie normale Pascal-Befehle. Ihr nicht vorhandener Rückgabewert darf keiner Variablen zugewiesen werden.

**Prozeduren sind Funktionen, die *nichts* zurückgeben.  
 Sie dürfen aber, wie Funktionen auch, Eingaben, Ausgaben, Verzweigen, Schleifen etc. haben.  
 Sie dürfen auch Parameter übernehmen und globale Variablen lesen.**

### 14.2.1 Modularisierung

Hauptprogramm:

```
[...]
BEGIN
  initialisiere;      (* zB Parameter parsen *)
  REPEAT
    zeigeMenue;
    bearbeiteWahl;
  UNTIL (wahl=0);
  raeumeAuf;         (* zB Dateien schließen *)
END.
```

### 14.3 Übung

Schreiben und testen Sie mit 1000 Zufallswerten ein Programm, das eine Funktion enthält, die das tut, was ihr Name sagt.

#### 14.3.1 Teste zwei INTEGER auf Gleichheit

Benutzen Sie folgende Syntax: `istGleich (...i...j...)....`

#### 14.3.2 Minimum von 5 INTEGER

Sie dürfen auch drei Funktionen schreiben: `min`, `durchschnitt`, `max`.

#### 14.3.3 Wertetabelle

Schreiben Sie die Funktion *sinxquadrat* =  $\sin(x^2)$ , und geben Sie eine Wertetabelle aus für  $x = 0$  bis 8:

x	sin(x)	cos(x)	sin(x*x)
---	--------	--------	----------

#### 14.3.4 Rekursiv: Fakultät

**Wiederholung:**  $0! = 1$ .  $n! = 1 * 2 * 3 * \dots * n$

Deshalb ist  $n! = 1$ , wenn  $n=0$  oder  $n=1$  ist. In allen anderen Fällen ist  $n! = n * (n-1)!$ .

ZB ist  $3! = 3 * 2! = 3 * 2 * 1! = 3*2*1 = 6$ .

TurboPascal verträgt Funktionen, die sich selber aufrufen. Können Sie sie programmieren?

#### 14.3.5 Größter Gemeinsamer Teiler

Ein Verfahren zum Bestimmen des GGT von a und b aus der griechischen Antike lautet:

- 1 wenn  $a < b$ , tausche sie;
- 2 wenn der Rest von  $a/b = 0$

dann ist b der GGT

sonst setze  $a = b$  und  $b =$  den Rest aus der Division und gehe nach 2

#### 14.3.6 BarbieMod

Zerlegen Sie "Barbie's Primzahltest" in mindestens drei Prozeduren.

#### 14.3.7 Quader4 in Prozeduren zerlegen

Rupfen Sie die Quaderberechnung in prozedurale Blöcke von max. 25 Zeilen Länge `init`, `menue`, `arbeite`, `quit` auseinander, schreiben Sie das Hauptprogramm dazu, zählen Sie seine Zeilen und überlegen Sie, wozu das gut sein soll.

### 14.3.8 Unlösbare Aufgabe - Newton- und Sekanten-Verfahren

$$-0.1 * x = \sin(x)$$

ist eine analytisch nicht lösbare Aufgabe.

Wir lösen sie numerisch mit hinreichender Genauigkeit, indem wir sie in eine Aufgabe zur Nullstellensuche umwandeln:

$$0 = 0.1 * x + \sin(x)$$

und dann die Nullstellen-Suchverfahren "Newton" und "Sekante" anwenden.

Gesucht ist die Lösung in der Nähe von 3.

a) Skizzieren Sie alle drei Graphen.

b) Schreiben Sie Struktogramm und Programm. Lagern Sie dabei  $f(x)$  in eine Funktion aus, und fordern Sie `startwert` und `delta` mit Defaultwerten vom User an.

Wenden Sie folgendes Verfahren an:

```
x <-- startwert
delta <-- 0,001
wiederhole
    f(x) * delta
    x <-- x - -----
                f(x+delta) - f(x)
bis genauigkeit groß genug
```

c) wie b), aber: Fordern Sie `x1` und `x2` mit Defaultwerten vom User an. Wenden Sie folgendes Verfahren an:

```
x1 <-- startwert1
x2 <-- startwert2
wenn ( f(x1) * f(x2) > 0 )
    dann ( vergiß es )
    sonst
    wiederhole
        f(x2) * (x2 - x1)
        x3 <-- x2 - -----
                    f(x2) - f(x0)
    wenn ( f(x3) * f(x2) > 0 )
        dann x2 <-- x3
        sonst x1 <-- x3
bis genauigkeit groß genug
```

### 14.3.9 Einstein

Wem gehört der Fisch? Lösen Sie die Frage EGAL WIE.

1. Es gibt 5 Häuser in 5 verschiedenen Farben.
  2. In jedem Haus wohnt eine Person mit einer anderen Nationalität.
  3. Jeder Hausbewohner bevorzugt ein bestimmtes Getränk, raucht eine bestimmte Zigarettenmarke oder hält ein bestimmtes Tier.
  4. Keine der fünf Personen trinkt das gleiche Getränk, raucht die gleichen Zigaretten oder hält das gleiche Tier wie einer seiner Nachbarn.
- Der Brite lebt im roten Haus.
  - Der Schwede hält einen Hund.
  - Der Däne trinkt gerne Tee.
  - Das grüne Haus steht links vom weißen Haus.
  - Der Besitzer des grünen Hauses trinkt Kaffee.
  - Die Person, die PallMall raucht, hält einen Vogel.
  - Der Mann im mittleren Haus trinkt Milch.
  - Der Besitzer des gelben Hauses raucht Dunhill.
  - Der Marlboro-Raucher wohnt neben dem, der eine Katze hält.
  - Der Mann mit dem Pferd wohnt neben dem Dunhill-Raucher.
  - Der Winfield-Raucher trinkt Bier.
  - Der Norweger wohnt neben dem blauen Haus.
  - Der Deutsche raucht Rothmann's.
  - Der Marlboro-Raucher hat einen Nachbarn, der Wasser trinkt.

## 15 Funktionen und Prozeduren - unsauber

### 15.1 Call by Value und Call by Reference

Bisher haben wir unseren Funktionen und Prozeduren die Parameter als Werte (value) übergeben. Änderungen der entsprechenden Variablen im Unterprogramm waren dadurch ausgeschlossen.

In Pascal gibt es jedoch die Möglichkeit, Variablen als *Variablen* (Referenz) zu übergeben. Mit diesen kann dann die Funktion bzw. Prozedur nach Belieben verfahren.

```
[...]
function haelfte (r:real):real;                (* sauber *)
begin
  haelfte := r/2;
end;

procedure halbiere ( VAR r : real) : real;      (* unsauber *)
begin                                           (* siehe ich übergebe dir eine real-variable *)
  r := r/2;                                     (* mach damit was du willst, und sag mir *)
end;                                           (* nichts davon *)

begin
  [...]
  s := haelfte (s);                             (* halbiert s ohne tricks *)
  halbiere (t);                                 (* halbiert t. Ob t dabei geändert wird,
                                             ist in der Syntax nicht sichtbar. *)
end.
```

**Beim "call by value " wird dem Unterprogramm der WERT einer Variablen übergeben. Das Unterprogramm kann diese Variable NICHT ändern.  
Beim "call by reference" wird dem Unterprogramm eine Variable übergeben. Das Unterprogramm kann diese Variable dann auch ändern - selbst wenn sie einen anderen Namen hat.**

### 15.2 Bewertung des Call by Reference

Der Call by Reference ist nur dann vorteilhaft, wenn man Variablen geändert haben möchte, zB inkrementiert, dekrementiert, vertauscht oder sortiert (für Felder).

Bei der Verwendung des CbR in Prozeduren ist immer deutlich zu erkennen, daß möglicherweise Variablen im Hauptprogramm geändert werden.

Bei der Verwendung des CbR in Funktionen ist NICHT immer deutlich zu erkennen, daß möglicherweise Variablen im Hauptprogramm geändert werden.

**Der Call by Reference soll in Funktionen nicht verwendet werden.**

#### 15.2.1 Übung: procedures tausche

Schreiben und testen Sie die zwei Prozeduren `tauschereal` und `tauscheint`.

### 15.3 Wie funktioniert Call by Value? [optional]

Der aufrufende Programmteil sitzt an irgendeiner Stelle im Hauptspeicher. Seine Variablen sind vom Compiler in Speicheradressen übersetzt werden.

Wenn ein Unterprogramm (Funktion oder Prozedur) aufgerufen wird, wirft das Hauptprogramm seine gegenwärtige Speicheradresse (die "Rücksprungadresse") sowie die Werte der Parameter auf einen Stack (Stapel, Kellerspeicher), der irgendwo im Speicher liegt, und springt an den Anfang des Unterprogramms.

Das Unterprogramm holt sich die Werte vom Stack und rechnet damit. Seine Variablen haben möglicherweise gleiche Namen wie im aufrufenden Programm, sitzen aber an anderer Stelle im Speicher und beeinflussen sich deshalb ÜBERHAUPT NICHT. Am Schluß holt es sich die Rücksprungadresse, wirft den Rückgabewert auf den Stack und springt zu der Rücksprungadresse.

An der Rücksprungadresse sitzt immer noch das aufrufende Programm. Es weiß noch, daß es eben gerade eine Funktion aufgerufen hat, holt sich den zurückgegebenen Wert vom Stapel und macht weiter wie gewohnt.

Dies funktioniert auch, wenn eine Funktion sich selbst aufruft. Viele rekursive Aufrufe einer Funktion bewirken dabei ein Anwachsen des Stacks - sonst nichts.

### 15.4 Wie funktioniert Call by Reference? [optional]

Das aufrufende Programm wirft nicht die Variablen-Werte, sondern die Variablen-Speicheradressen auf den Stapel. Beides darf gemischt werden.

Aufrufendes und aufgerufenes Programm wissen, in welcher Reihenfolge die Adressen und Werte abgelegt wurden und wieder geholt werden müssen.

### 15.5 Didaktische Bewertung der Calls by XXX bzw. Global-Lokal

1. Wenn ich ein Thema anfasse, mache ichs ganz. Oder gar nicht.
2. Diese beiden Themen beinhalten einen Berg von Theorie. Für Unterricht und Übung muß man mindestens drei Doppelstunden ansetzen.
3. Sie sind geeignet, die Prozeduren `tausche`, `inc`, `dec` und `sort` (mit Übergabe eines Feldes by reference) elegant schreiben zu können.
4. Aber: `inc` und `dec` sind bereits in Pascal eingebaut.
5. Aber: `tausche` drückt sich um einen Dreizeiler: `h:=a; a:=b; b:=h;`.
6. Aber: Felder und Sortieralgorithmen sind kompliziert genug, Call by XXX packt nur zusätzliche Schwierigkeiten obendrauf.

**Fazit** Ich halte das Kapitel "Funktionen - sauber" für sinnvoll und rund, aber das Kapitel "Call by XX - Global-Lokal" für überladen und unnütz. Punkt.

## 15.6 Globale und lokale Variablen

Problem: Jemand hat die Variablen in seiner Funktion schlampig definiert:

```

program p;
var i,j,k : integer;

function myinc (i : integer) : integer;
begin
  j := j + 1;
  myinc := i + 1;
end;

begin
  j := 0;
  k := 1;
  k := myinc (k);
  writeln (i:8, j:8, k:8);
end.

```

In der Funktion `myinc` wird die Variable `i` definiert. Sie bekommt den Wert von `k` (also 1) übergeben und gibt 2 zurück. Soweit ist alles in Ordnung.

In der Funktion `myinc` ist die Variable `j` NICHT definiert. Der Compiler sieht aber die Variable `j` des Hauptprogramms und benutzt diese - mit überraschenden Ergebnissen.

**Variablen, die innerhalb eines Unterprogramms definiert sind, heißen "lokal".  
Variablen, die im Hauptprogramm definiert sind, heißen "global".**

**Wenn ein Unterprogramm lokale Variablen hat, kann es die globalen Variablen gleichen Namens nicht sehen.  
Aber: globale Variablen, die nicht re-definiert worden sind, kann es jederzeit sehen und ändern.**

**Unterprogramme, die globale Variablen *lesen*, sind handlich.  
Unterprogramme, die globale Variablen *schreiben*, sind unübersichtlich und schwer wartbar.  
Definieren Sie alle Variablen, die Sie in jeder Prozedur einsetzen, sonst wird aus Versehen eine globale Variable geändert.**

## 16 Zufallszahlen und Konstanten

### 16.1 Zufallszahlen

```
randomize mischt den Zufallszahlengenerator neu.  
randomize sollte am Programmanfang stehen.  
random gibt eine Zufallszahl vom Typ REAL zwischen 0 und 1 zurück. Maximal-  
Wert=0.9999999999.  
random (12) gibt eine Zufallszahl zwischen 0 und 11 zurück.  
a + random(b) gibt eine Zufallszahl zwischen a und a+b-1 zurück.
```

### 16.2 Konstanten

**Konstanten definiert man vor den Variablen wie folgt:**

```
CONST  
  n = 16;          (* eine INTEGER-Konstante *)  
  r = 14.0;       (* eine REAL-Konstante *)  
  doofi = TRUE;   (* ... *)
```

**Konstanten sind so konstant, daß sie sogar für Feldbegrenzer taugen:**

```
CONST  
  myn = 16;       (* eine INTEGER-Konstante *)  
VAR  
  myf : ARRAY [-myn .. myn] OF REAL;   (* 33 Elemente *)
```

## 17 Felder

Stellen Sie sich vor, sie haben 5 INTEGER und sollen mit diesen etwas machen. ZB Summe, Minimum, Durchschnitt, Maximum, quadratische Abweichung berechnen.

Das macht viel Arbeit: 5 Variablen definieren, und dauernd ganze Programm-Blöcke kopieren, einfügen und einen oder zwei Variablennamen ändern.

Wenn es plötzlich 6 INTEGER sein sollen, wird das Programm vermutlich doppelt so groß.

Wenn es 50 oder 5000 INTEGER werden sollen, ist das mit unseren bisherigen Mitteln nicht zu schaffen.

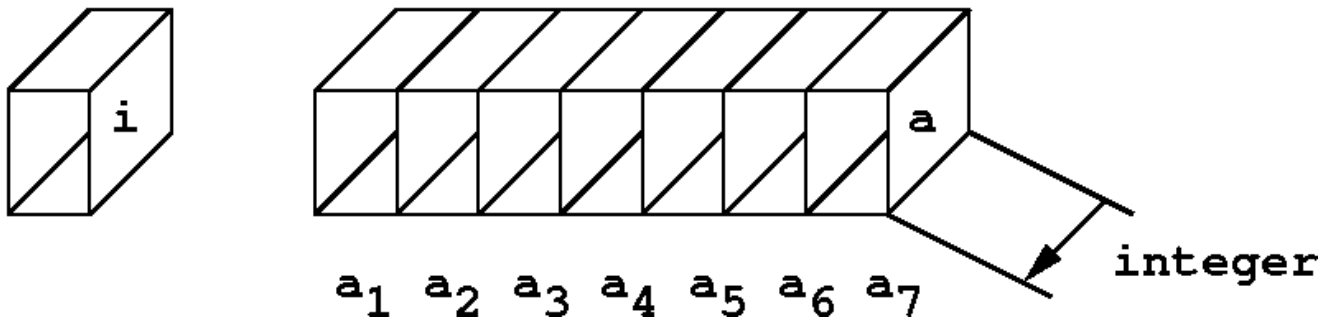
**Dabei versteht doch jedes Kind:** "Zähl doch bitte mal den 1. bis 5. INTEGER zusammen."

TurboPascal versteht's auch.

**Ein 1-dimensionales Feld ist eine hohe Kommode.**

**Die Kommode hat einen Namen, und jedes Schubfach hat eine Nummer vom Typ INTEGER.**

**Die Schubfächer sind alle gleich tief, so daß Daten eines Typs hineinpassen.**



Natürlich können wir nicht mit dem ganzen Feld rechnen, aber mit den Daten aus den einzelnen Schubladen. Diese verhalten sich genauso, wie es ihr Datentyp verspricht.

"He Turbopascal! Ich definiere eine Kommode – err, ein Feld namens f. Die Schubladen – err, die Nummern laufen von -2 bis +3. Und sie sind vom Typ INTEGER.

Und noch ein i."

```
VAR
  f : array [-2 .. +3] OF INTEGER;
  i : INTEGER;
```

"Ich weise der Schublade mit der Nummer 0 den Wert 5 zu."

```
f[0] := 5;
```

"Ich gucke nach, ob du das auch richtig gemacht hast."

```
WRITELN (f[0]);
```

"Kannst du damit auch rechnen?"

```
i := 17;
WRITELN (f[0] * f[0]);
WRITELN (i * f[0]);
```

"Dein Index ist INTEGER. i ist INTEGER. Friß dies:"

```
i := 0;  
WRITELN (f[i]);
```

“Jetzt null ich dich aus.”

```
FOR i := -2 to 3 do BEGIN  
  f[i] := 0;  
END;
```

“Zeig wer du bist.”

```
FOR i := -2 to 3 DO BEGIN  
  WRITELN ('In Schublade ', i, ' steckt ', f[i]);  
END;
```

**Ein 1-dim Feld ist eine Reihe gleichartiger Daten (BOOLEAN, REAL, ...).**  
**Wenn man es definiert, kann man auf die einzelnen Elemente unter Angabe des Feldnamens und der lfd. Nummer zugreifen.**  
**Mit FOR-DO-Schleifen o.ä. kann man in einem Rutsch auf alle Elemente zugreifen.**

## 17.1 Übung

### 17.1.1 Mein erstes Feld

Definieren Sie ein Feld namens `me f`, `Index=0..5`, `Typ=INTEGER`.

Schreiben Sie in alle Elemente 0 hinein und geben Sie sie aus.

Schreiben Sie in alle Elemente  $i*i$  hinein und geben Sie sie aus.

Schreiben Sie in alle Elemente  $100-i*i$  hinein und geben Sie sie aus.

Pause.

Schreiben Sie in alle Elemente eine Zufallszahl hinein und geben Sie sie aus.

### 17.1.2 Summe, Minimum, Mittelwert, Maximum

Definieren Sie ein Feld namens `f` mit `n REAL` Elementen, `n=10`.

Schreiben Sie überall Zufallswerte hinein, und geben Sie's aus.

Setzen Sie `sum` auf 0, laufen Sie einmal durch das Feld, machen Sie dabei *irgendwas*, und präsentieren Sie die Summe aller Elemente.

Setzen Sie `min` auf `f[1]`, setzen Sie `max` auf `min`, und präsentieren Sie den Rest.

Setzen Sie `n` auf 999, und schauen Sie, ob's noch geht.

### 17.1.3 Lottozahlen

Geben Sie 7 verschiedene Zahlen von 1 bis 49 aus.

Hinweis: Benutzen Sie ein `array of boolean`.

### 17.1.4 Bubblesort

Erzeugen Sie die Konstante `n=20`. Erzeugen Sie ein Feld von `1..n` von Typ `integer`, schreiben Sie in jede Zelle eine Zufallszahl zwischen 0 und 99, und geben Sie es in einer Zeile aus.

**Algorithmus** Lassen Sie `i` `n`-mal durch das Feld laufen. Wenn der `i`. Wert größer ist als sein rechter Nachbar, tauschen Sie die beiden.

Geben Sie anschließend das Feld nochmal aus.

a) Schreibtischtest für `n=5`.                      b) Struktogramm                      c) Programm                      d) `n=2000`.

### 17.1.5 MinSort

Aufgabe s. Bubblesort.

**Algorithmus** Laufen Sie durch das gesamte Feld, finden Sie das Minimum, und merken Sie sich, wie groß es ist und wo es steht. Tauschen Sie es anschließend mit dem ersten Element.

Laufen Sie durch das gesamte Feld ohne das erste Element, finden Sie das Minimum, und merken Sie sich, wo es steht. Tauschen Sie es anschließend mit dem zweiten Element.

Usw.