Kleine Einführung in Turbo-Pascal

Die Programmiersprache **PASCAL** wurde 1970 durch Niklaus Wirth an der ETH Zürich entwickelt. Sie wurde als eine Sprache entworfen, die zur Ausbildung von Programmiererinnen und Programmieren geeignet ist. Die Sprache ist einfach und klar strukturiert und so konzipiert, dass sie auf „kleinen“ Computern eingesetzt werden kann. Namensgeber für die Sprache PASCAL war der französische Mathematiker Blaise Pascal, der im Jahr 1652 einen der ersten mechanischen Rechenautomaten erfunden hat.
Seit 1985 gab es auf DOS-Rechnern den Sprachstandard **Turbo-Pascal** 3.0, der in den Folgejahren über die Versionen 4 - 5.5 - 6 - 7 ständig weiter entwickelt wurde. Die Windows-Variante von PASCAL ist DELPHI und sie erfreut sich heute (2002) zunehmender Beliebtheit.

Das Ziel bei der Entwicklung war es, eine Sprache zu erfinden, die:

- auf wenigen, grundlegenden Konzepten aufbauend
- einfache, übersichtliche Sprachkonstruktionen und
- einfache sprachliche Regeln (Syntax) aufweist und
- mit einem einfachen Compiler effizienten Code erzeugt.

Die folgenden Programmbeispiele wurden alle mit der Version Turbo-Pascal 5.5 erstellt. Diese Entwicklungsumgebung kann inzwischen als sog. "freeware" kostenlos genutzt werden. Unter WIN 95/98 sollte auf dem Desktop eine Verknüpfung zum Programm "turbo.exe" erstellt werden und unter "Eigenschaften–Bildschirm" Fenster eingestellt werden.

§ 1 Lineare Programme

Die einfachsten Programme bestehen im wesentlichen aus einem **E**ingabeteil, einem **V**erarbeitungsteil und einem **A**usgabeteil (EVA-Prinzip). Die einzelnen Teile werden der Reihe nach ausgeführt. Eine solche Aneinanderreihung von Anweisungen heißt **Sequenz**.

Beispiel 1: Die Sekretärin einer Firma soll aus dem Nettobetrag die Mehrwertsteuer und den Bruttobetrag berechnen. Sie möchte den Computer diese Arbeit ausführen lassen und schreibt dazu ein Programm, d.h. sie formuliert alle erforderlichen Schritte in einer Programmiersprache, so dass ein Computer die Anweisungen versteht und ausführen kann.

Der Programmtext wird in einem sog. **Editor** geschrieben. Dabei stehen u.a. folgende Befehle über die Tastatur zur Verfügung:

- Ctrl-Y : löscht eine Zeile
- Ctrl-N : schafft eine neue Leerzeile
- F10 in die Menüzeile
- F2 : Speichern des Textes
- F3 : Laden eines Textes
- F10 C : Übersetzen des Programms
- F10 R : Starten des Programms

In Pascal müssen alle Variablen zunächst mit VAR vereinbart werden. Dabei können Variablen von verschiedenem Typ vereinbart werden.

- **real :** Kommazahlen von $-2{,}9 \cdot 10^{39}$ ... $+1{,}7 \cdot 10^{38}$
- **integer :** ganze Zahlen (von -32767 bis 32767)
- **LONGINT**: ganze Zahlen (von –2147483648 bis 2147483647)
- **char :** einzelnes Zeichen
- **string :** Worte mit einer maximalen Länge von 255 Zeichen

```
PROGRAM Rechnung;
uses crt;

var nettobetrag, bruttobetrag, mwst : REAL;

BEGIN TextBackground(blue);
  TextColor(yellow);
  ClrScr;
  WriteLn('   Rechnung mit Mehrwertsteuer');
  WriteLn('   ============================');
  WriteLn;

  {Eingabeteil}
  Write('   Nettobetrag = ');
  ReadLn(nettobetrag);

  {Verarbeitungsteil}
  mwst := nettobetrag*15/100;
  bruttobetrag := nettobetrag + mwst;

  {Ausgabeteil}
  WriteLn;
  WriteLn('   Mehrwertsteuer = ',mwst:12:2);
  WriteLn('   Bruttobetrag   = ',bruttobetrag:12:2);
  ReadLn;
END.
```

Das Programm zwischen BEGIN . . . END.“ wird auch als **Hauptprogramm** bezeichnet . Durch die Anweisung "**uses crt;**" stehen einige Befehle zur Gestaltung der Bildschirmausgabe zur Verfügung. CRT=cathode ray tube (Elektronen-strahlröhre)

ClsScr; löscht den Bildschirm.
TextColor (farbe); legt die Textfarbe fest.
TextBackground (farbe); bestimmt die Farbe des Bildschirmhintergrundes.
Beliebt ist z.B. die Kombination textbackground(yellow); textcolor(blue);
n := ReadKey; Eingabe nur eines einzigen ASCII-Zeichens ohne Return
GotoXY(x,y); Cursor an die Bildschirmposition x,y positionieren
Readln(x); liest eine Variable ein.
Writeln(x); Ausgabe in eine neue Zeile
Write(x); Ausgabe an aktueller Cursorposition
WhereX; Cursorspalte ermitteln
WhereY; Cursorzeile ermitteln
ReadKey; Tastatur abfragen

Anm.: Damit die Ausgabe am Ende des Programms auf dem Bildschirm bleibt, sollte am Ende die Anweisung "Readln;" stehen. Das Programm endet dann erst nach Betätigen der Return-Taste.

Aufg.: Schreibe ein Programm, das nach Eingabe von Länge und Breite eines Rechtecks in Metern den Umfang und den Flächeninhalt ausgibt!

Ähnlich wie in Windows kann man im DOS-Editor von Turbo-Pascal Programmteile markieren und kopieren. Einige wichtige Editierkommandos:

Zeile löschen Ctrl+Y
Zeile einfügen Ctrl+N
Suchen Ctrl+Q-F
Blockanfang markieren Ctrl+K-B
Blockende markieren Ctrl+K-K
Block entmarkieren Ctrl+K-H
Block kopieren Ctrl+K-C
Block verschieben Ctrl+K-V
Block löschen Ctrl+K-Y
Block lesen (File) Ctrl+K-R
Block schreiben (File) Ctrl+K-W

Hotkeys

Hilfestellung F1
Programmtext sichern F2
Programmtext laden F3
Fensterwechsel F6
Schließen eines Fensters Alt+F3
User-Screen Alt+F5
Aktiviert Syntaxhilfe Ctrl+F1
Compile Alt+F9
Run Ctrl+F9

Die **Ausgabeanweisung**

Das Schlüsselwort zur Ausgabe von Daten auf den Bildschirm lautet zunächst **Write** oder **Writeln**. Dahinter wird in runden Klammern angegeben, was ausgegeben werden soll. Hier unterscheidet man verschiedene Möglichkeiten :

Ausgabe von Textkonstanten	Ausgabe einer Variablen
Writeln (' Text ');	**Writeln (Variable);**

Beachte, dass Texte zwischen Hochkommata stehen müssen, um sie von den Namen der deklarierten Variablen abzugrenzen. Will man in einer Anweisung mehrere Elemente gleichzeitig ausgeben wollen, so müssen die einzelnen Komponenten in der Ausgabeanweisung durch Kommata getrennt werden.

Beispiel : Writeln (' Die Entfernung beträgt : ' , Laenge, ' km ');

Ausgabeformatierung

Damit Gleitkommazahlen (Realzahlen) am Bildschirm nicht mit 10 Stellen hinter dem Komma und mit Exponent zur Basis 10 ausgegeben werden (das ist die wissenschaftliche Schreibweise), wird die Ausgabeanweisung durch Parameter ergänzt, die eine Fließkomma-Notation ermöglichen. Das heißt, hinter dem Variablennamen, jeweils durch Doppelpunkte getrennt, werden die Gesamtzahl aller gewünschten Stellen - inklusive Dezimalpunkt und Vorzeichen - und die Nach-Stellen hinter dem Dezimalpunkt angeben.

Beispiel:

Art	Befehl	Ergebnis
nichtformatierte Ausgabe	WriteLn (Strom);	1.3421645E-02
formatierte Ausgabe	WriteLn (' Ibe =', Strom:5:1, ' mA');	Ibe = 13.4 mA

Aufgabe:

In Amerika wird die Temperatur nicht in Grad Celsius, sondern in Grad Fahrenheit gemessen. Es gilt : 0 °C ≡ 32 °F; 100 °C ≡ 212 °F
Schreibe ein Programm, das nach Eingabe einer Temperatur in Celsius diese in Fahrenheit ausgibt!

Standardfunktionen

Wie in allen Programmiersprachen, so sind auch in Turbo-Pascal viele mathematische Funktionen bereits vordefiniert.

- y := Sqr (n); Quadratfunktion
- y := Sqrt (n); Wurzelfunktion (square root)
- y := Exp(n); Exponentialfunktion zur Basis e (e = 2,718281..)
- y := Sin(n); Sinusfunktion (Ergebnisse liegen im Bogenmaß vor !)
- y := Cos(n); Cosinusfunktion .. ebenfalls im Bogenmaß
- y := ArcTan(n); Umkehrfunktion des Tangens (n in rad !)
- y := Trunc(n); Nachkommastellen abschneiden
- y := Round(n); Wert runden
- y := Abs(n); Absolutbetrag (Vorzeichen spielen keine Rolle mehr)
- y := Ln (n); Logarithmus naturalis (Basis = e)
- y := UpCase (n); Buchstaben in Großbuchstaben umwandeln
- y := Random(n); ergibt eine Zufallszahl zwischen 0 und n

Beispiel 2: Schreibe ein Programm, das nach Eingabe der Kathetenlängen eines rechtwinkligen Dreiecks die Länge der Hypothenuse berechnet!

```
PROGRAM dreieck; {berechnet die Hypothenuse eines
rechtwinkligen Dreiecks}
uses crt;
var a,b,c : REAL;

BEGIN TextBackground(blue); TextColor(yellow);
  ClrScr;
  WriteLn;
  WriteLn(' Hypotenuse eines rechtwinkl. Dreiecks');
  WriteLn;
  Write('    Länge der 1.Kathete :   ');
  ReadLn(a);
  Write('    Länge der 2.Kathete :   ');
  ReadLn(b);
  c := Sqrt ( a*a + b*b);
  WriteLn;
  WriteLn('    Länge der Hypotenuse: ',c:10:4);
  ReadLn;
END.
```

§ 2 Bedingte Anweisungen

Zu den wohl wichtigsten Kontrollstrukturen gehören die ”bedingten Anweisungen". Hiermit ist es möglich, die weitere Programmausführung von einer (oder mehreren) Bedingungen abhängig zu machen.
Der grobe Aufbau ist wie folgt (der ELSE-Zweig kann auch entfallen):

```
IF Bedingung THEN BEGIN
[...Bedingung ist erfüllt...];
END
ELSE BEGIN
[...Bedingung ist nicht erfüllt...];
END;
```

Vergleichsoperatoren: < > = <= >= <>

Man kann mehrere Bedingungen mit ” logischen“ Verknüpfungen zusammenfassen. Hierzu stehen in Turbo-Pascal folgende Operatoren zur Verfügung:

AND OR NOT

Beispiel:

IF (Bedingung1) AND (Bedingung2) THEN ...

Ein ” BEGIN ...END;“ kann in einer Struktur mit nur einer Anweisung entfallen! Es darf kein Semikolon nach einem Befehl direkt vor einem ELSE-Zweig gesetzt werden!

Beispiel 3: Ein Programm soll den Warenwert einlesen. Ist dieser größer als 1000 Euro, so wird ein Rabatt von 10 % gewährt. Anschließend werden noch 16 % Mehrwertsteuer hinzu berechnet.

```
PROGRAM Rabatt_Berechnung;
uses crt;
var warenwert, nettowert, rabatt, mwst, bruttowert :
REAL;

BEGIN TextBackground(blue); TextColor(yellow);
  ClrScr;
  WriteLn;
  WriteLn('  Warenwert mit Rabatt und Mehrwertsteuer');
  WriteLn;
  Write('    Warenwert :   '); ReadLn(warenwert);
  IF warenwert >= 1000 THEN
    rabatt := warenwert*0.10
```

```
  ELSE rabatt := 0;
  nettowert := warenwert - rabatt;
  mwst := nettowert * 0.16;
  bruttowert := nettowert + mwst;
  WriteLn;
  WriteLn('    Warenwert     : ',warenwert:10:2);
  WriteLn('    Rabatt        : ',rabatt:10:2);
  WriteLn('    Nettowert     : ',nettowert:10:2);
  WriteLn('    Mehrwertsteuer: ',mwst:10:2);
  WriteLn('    Bruttowert    : ',bruttowert:10:2);
  ReadLn;
END.
```

Aufgaben:

1. Ein Programm liest die drei Seitenlängen eines Dreiecks ein und entscheidet, ob das Dreieck gleichseitig oder gleichschenklig ist.

2. Bei einer Schadensversicherung hat der Versicherte bei Schäden über 100 Euro 20 % der Schadenssumme, mindestens jedoch 100 Euro selbst zu tragen. Erstelle ein Programm, das nach Eingabe der Schadenssumme den Anteil des Versicherten und die Zahlung der Versicgerung ausgibt!

3. Schreibe ein Programm, das eine quadratische Gleichung in der Normalform $x^2 + px + q = 0$ löst! Dazu werden die beiden Variablen p und eingelesen, die Diskriminante berechnet und entschieden, ob es 0, 1 oder 2 Lösungen gibt. Sinnvollerweise sollten existierende Lösungen auch ausgegeben werden. Zerlege das Programm in mehrere Prozeduren!

§ 3 Wiederholungen

3.1 Wiederholung mit FOR ... TO (DOWNTO) ...DO

Die FOR-Schleife ist eine Zählschleife. Die Zählvariable wird zunächst mit einem Startwert initialisiert. Anschließend werden die Anweisungen innerhalb der Schleife so oft wiederholt, bis der Zähler den Endwert erreicht hat. Dabei kann vorwärts oder rückwärts gezählt werden, aber immer nur in Einer-Schritten.

Beispiel: Dieses kleine Programm gibt die ersten 20 Quadratzahlen aus.

```
PROGRAM for_schleife;
USES Crt;
VAR i:INTEGER;
    BEGIN
    ClrScr;
    FOR i:=1 TO 20 DO
    WriteLn(i, '   ',i*i);
    ReadLn;
END.
```

3.2 Wiederholung mit REPEAT Anweisungen UNTIL Bed

Die Anweisungen innerhalb der REPEAT-Schleife werden solange wiederholt, bis die UNTIL-Bedingung wahr ist, also den Wert TRUE liefert. Diese Schleife wird mindestens einmal ausgeführt, da die Prüfung erst am Ende der Schleife geschieht. Die Übersetzung dieser Schleife lautet also: ” Wiederhole solange, bis die Bedingung Wahr ist“.

Beispiel: Das Programm wartet, bis eine beliebige Taste gedrückt wurde. Erst dann wird die nächste Anweisung ausgeführt.

```
PROGRAM repeat_schleife;
   USES Crt;
   BEGIN ClrScr;
   repeat
   until readkey <>'';
   writeln(' Es geht weiter.');
   ReadLn;
END.
```

3.3 Wiederholung mit WHILE <Bed> DO <Anw>

Die WHILE-Schleife ist das Gegenstück zur REPEAT-Schleife. Sie wird ausgeführt, solange die Durchführungs-Bedingung wahr ist. Die Prüfung geschieht am Schleifenanfang, d.h. ggf. wird die Schleife erst gar nicht ausgeführt.

Beispiel: Nach dem Euklid'schen Algorithmus wird der ggT, also der größte gemeinsame Teiler zweier Zahlen ermittelt. Solange die beiden Zahlen noch verschieden sind, wird die größere Zahl durch die Differenz beider Zahlen ersetzt. Sobald die Zahlen gleich sind, hat man den ggT gefunden.

```
PROGRAM while_scheife;
USES Crt;
var a,b : INTEGER;
BEGIN
  ClrScr;
  a := 56; b:= 64;
  while a <> b DO
    BEGIN IF a>b THEN a := a-b
    ELSE b := b-a;
    END;
  WriteLn(' ggT = ', a);
  ReadLn;
END.
```

Aufgaben: 1. Vom Schaum in einem Bierglas zerfällt pro Minute 30 %. Nach welcher Zeit hat sich der Schaum auf 1 % der Ausgangsmenge reduziert?

2. Schreibe ein Programm zur Ausgabe der Quadratwurzeln der natürlichen Zahlen von 1 bis 20! Verwende zuerst eine repeat-, dann eine while-Schleife!

2. Es soll eine Tabelle mit dem kleinen 1x1 auf dem Bildschirm angezeigt werden.
Tipp: Verwende zwei ineinander geschachtelte for-Schleifen!

```
 1  2   3   4   5   6   7   8   9  10
 2  4   6   8  10  12  14  16  18  20
 3  6   9  12  15  18  21  24  27  30
. . . . . . . . . . . . . . . . . . .
10 20  30  40  50  60  70  80  90 100
```

3. Drei natürliche Zahlen a, b und c heißen Pythagoras-Zahlen, wenn für die Zahlen gilt: $a^2 + b^2 = c^2$. Schreibe ein Programm, das systematisch nach solchen Zahlen sucht!

4. Programmiere ein Zahlenratespiel. Der Computer ermittelt eine natürliche Zahl von 1 bis 100, die der Anwender erraten muss.

Hinweis: Eine Zufallszahl von 1 bis 100 wird ermittelt, indem einmal im Programm die Anweisung randomize; aufgerufen wird. Die Anweisung
y := random(100) + 1; ergibt dann die Zufallszahl.

5. Schreibe ein Programm, das die Wertetabelle einer Funktion f mit
$f(x) = x^3 - 2x + 1$ für $-4 \leq x \leq 4$ ausgibt. Der x-Wert soll dabei in Schritten von 0,5 erhöht werden. Löse die Aufgabe mit einer repeat- und einer while-Schleife!

6. Im Nordpolarmehr bei Kanada leben nach Schätzungen der Wissenschaftler zur Zeit 2600 Kegelrobben, die sich jährlich um 7,5 % vermehren. 250 Jungrobben werden

aber jedes Jahr wegen ihres Felles getötet. Entwickle ein Programm, das die Entwicklung des Robbenbestands für die nächsten 10 Jahre berechnet und die Zahlen tabellarisch geordnet am Bildschirm ausgibt!

7. Herr Spar legt ein Kapital für 6 Jahre zu einem festen Zinssatz mit Zinseszins an. Wie entwickelt sich sein Kapital im Laufe der 6 Jahre?

8. Auf einer Wiese hütet Liesel Gänse und anderes Kleinvieh. Der Technofreak, der gerade vorbeikommt, fragt Gänseliesel nach der Anzahl ihrer Tiere. Liesel antwortet: Es sind doppelt so viele Gänse wie Hühner, aber dreimal so viele Kaninchen wie Schafe. Der Technofreak zögert und zählt bei den Tieren insgesamt 90 Beine. Mit diesen Angaben schreibt er zu Hause ein Programm, das ihm angibt, wie viele Tiere in welcher Gattung Liesel hütet. Schaffst du das auch?

§ 4 Prozeduren

Ein langes Programm kann sehr schnell unübersichtlich werden. Man fasst deshalb Teilprobleme zu Unterprogrammen zusammen. Aus dem Hauptprogramm werden diese **Unterprogramme** dann entsprechend aufgerufen. Das bringt vor allem dann Vorteile, wenn ein Unterprogramm mehrfach genutzt werden kann. Hier einige der von Pascal mitgelieferten Standardprozeduren : ClrScr; GotoXY(x,y); Randomize; Delay(x) usw.

1. einfache parameterlose Prozeduren

Unterprogramme müssen im Hauptprogramm zunächst deklariert werden. Das geschieht mit dem Schlüsselwort PROCEDURE gefolgt vom selbst gewählten Namen des Unterprogramms und dem Semikolon. Der Anweisungsteil der Prozedur steht genau wie beim Hauptprogramm zwischen BEGIN und END; hier allerdings gefolgt von einem Semikolon !

Beispiel:

```
PROCEDURE warte;
   BEGIN
     Repeat
     Until readkey <>'';
   END;
```

Im Hauptprogramm wird das Unterprogramm einfach mit seinem Prozedurnamen, hier **warte;** aufgerufen. Allgemein stellt der Aufruf einer Prozedur eine Anweisung dar.

2. Lokale und globale Variable in Prozeduren

Prozeduren sind fast genau so aufgebaut wie Programme selbst. Sie können deshalb auch einen eigenen Deklarationsteil enthalten. Variable, die innerhalb der Prozedur deklariert worden sind, gelten nur innerhalb dieser Prozedur. Im übergeordneten Hauptprogramm sind sie unbekannt (**lokale Variable)**. Im Hauptprogramm deklarierte Variable sind im ganzen Programm, also auch in den Unterprogrammen gültig. Werden sie im Unterprogramm verändert, so haben sie auch im Hauptprogramm geänderte Werte. Es können also **unerwünschte Seiteneffekte** entstehen. Man nennt solche Variablen **globale Variable**.

3. Prozeduren mit Parameterübergabe

Interessant ist die Möglichkeit, einer Prozedur Werte zu übergeben. Dazu wird der Prozedurkopf um Namen und Datentypenbezeichnung der empfangenen Variablen (Eingangsparameter) ergänzt.

Beispiel : PROCEDURE Addiere (x, y : integer);

Oft werden aus den Eingabeparametern neue Werte berechnet, die als Variable wieder an das Hauptprogramm übergeben werden. Die entsprechenden Ausgabeparameter müssen im Prozedurkopf durch ein vorangestelltes **VAR** im Prozedurkopf gekennzeichnet werden.

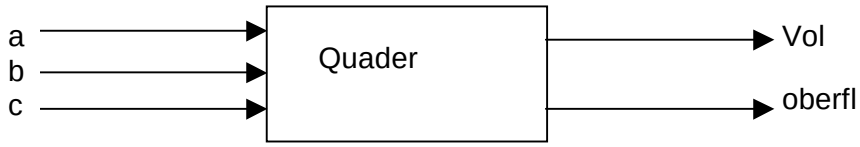
Beispiel: Eine Prozedur Quader soll als Eingabeparameter Länge, Breite und Höhe eines Quaders erhalten und daraus das Volumen und die Oberfläche berechnen und als Ausgabeparameter übergeben.

```
PROGRAM prozedur_demo;
USES Crt;
var laenge, breite, hoehe,vol, oberfl : REAL;

PROCEDURE quader(a,b,c:REAL; var V,OF:REAL);
{Prozedur zum Quader}
BEGIN V := a*b*c;
  OF := 2*a*b + 2*b*c + 2*a*c;
END;

BEGIN TextBackground(blue); TextColor(yellow);
  ClrScr;
  WriteLn('   Quaderberechnung');
  quader(2,2,3,vol,oberfl);
  WriteLn('   Volumen    : ',vol:12:2);
  WriteLn('   Oberfläche : ',Oberfl:12:2);
  REPEAT
  until ReadKey <>'';
END.
```

Schema der Prozedur:



Eingabeparameter können Zahlen oder Variablen sein, die Ausgangsparameter müssen Variable sein!

Aufgaben: Teste alle Prozeduren jeweils mit in einem kleinen Hauptprogramm!

1. Schreibe eine Prozedur **sterne**, die als Eingabeparameter eine natürliche Zahl n erhält und die dann n Sterne * in eine Zeile schreibt.

2. Schreibe eine Prozedur **eingabe(x,u,o)** mit dem Ausgangsparamater x und den Eingangsparametern o und u. Die Prozedur soll eine reelle Zahl einlesen, die aber zwischen den Grenzen u und o liegen muss. Eventuell muss die Eingabe mehrmals wiederholt werden!

3. Schreibe eine Prozedur **ggt(a,b,g)** mit den Eingangsparametern a und b (natürliche Zahlen). Die Prozedur ermittelt den ggT der Zahlen a und b und übergibt diesen an den Ausgangsparameter g.

4. Schreibe eine Prozedur **tausche(a,b)**, welche die Inhalte der beiden Zahlen a und b vertauscht. Wie müssen die Parameter a und b deklariert werden, damit es klappt?

5. (zum Knobeln) Eine Firma möchte insgesamt 521 Mitarbeiter zu einem längeren Wochenende an ein gemeinsames Ausflugziel transportieren lassen. Dem Busunternehmer stehen dazu drei Reisebusgrößen zur Verfügung : Busse mit 21 Sitzplätzen; Busse mit 41 Sitzplätzen und Busse mit 43 Sitzplätzen.

a) Gesucht ist zunächst die optimale Auslastung (Besetzung) der verschiedenen Bustypen; das heißt : In welcher Kombination werden die Busse für den Transport eingesetzt, damit keine Sitzplätze frei bleiben?

b) Da mehrere gleichwertige Lösungen in Frage kommen, stellt sich die Zusatz-aufgabe, welche Lösung die kostengünstigste ist. Suche diese Lösung, wenn der Einsatz der Busse mit 300.- €, 420.- € und 500.- € (in gleicher Reihenfolge) kalkuliert werden muss. Das Computerprogramm soll auch hier helfen.
(*Lösung : 5560.- DM für die Kombination 1 Bus a' 21 Plätze; 8 Busse a' 41 und 4 Busse a' 43 Plätze)*

§ 5 Funktionen

Funktionen sind spezielle **Prozeduren**, die als Ergebnis genau einen einzigen Wert, den Funktionswert, an das aufrufende Programm zurückliefern. Auch sie müssen im Deklarationsteil des Hauptprogramms vereinbart werden. Das Schlüsselwort lautet **FUNCTION** gefolgt von dem Namen und der formalen Parameterliste. Zusätzlich wird hinter einem Doppelpunkt immer ein **Datentyp** als Ergebnistyp erwartet.

Beispiel: Die in der Mathematik üblichen Funktionen können wie folgt vereinbart werden.

```
PROGRAM function_demo;
USES Crt;
var x: REAL;

FUNCTION f(x:REAL):REAL;
BEGIN f := x*x - 4;
END;

BEGIN TextBackground(blue); TextColor(yellow);
  ClrScr;
  WriteLn('   Wertetabelle');
  WriteLn;
  x := -5;
  REPEAT WriteLn(x:10:2,'   / ',f(x):10:2);
    x := x+1;
  until x >= 5;
  REPEAT
  until ReadKey <>'';
END.
```

Der Vorteil von Funktionen gegenüber Prozeduren besteht darin, dass sie im Hauptprogramm wie ein Variable behandelt werden und nicht wie eine Prozedur als Anweisung. Deshalb können nur Funktionen direkt in allen Ausdrücken z.B. Ausgabeanweisungen aufgerufen werden, ohne den Ausgabewert zuvor einer Variablen zuweisen zu müssen.

⇒ **Der Funktionswert muss am Ende der Funktion dem Funktionsnamen zugewiesen werden.**

Aufgaben: Teste alle Funktionen jeweils in einem kleinen Hauptprogramm!

1. Schreibe eine Funktion **quader**, die aus Länge, Breite und Höhe das Volumen berechnet!

2. Schreibe eine Funktion **ggT(a,b)**, die den ggT der beiden Zahlen a und b nach dem früher beschrieben Euklid'schen Algorithmus ermittelt!

3. Schreibe eine Funktion **zins(K,p,i)**, die aus dem Kapital K, dem Zinssatz p und der Anzahl i der Jahre (das können auch Bruchteile sein) die Zinsen berechnet!

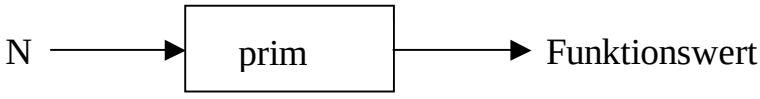
4. Schreibe eine Funktion **fak(n)**, um die Fakultät einer natürlichen Zahl n zu berechnen! Beachte: 0! = 1! =1.
Anm: Neben der iterativen Lösung mit einer Zählschleife ist auch die rekursive Lösung möglich.

5. In Pascal fehlt eine Funktion zum Potenzieren. Schreibe eine Funktion **pot(x,n)**, die zu einer reellen Zahl x und einer natürlichen Zahl n die Potenz x^n berechnet!

6. Die folgende Funktion **prim** testet auf sehr einfache Weise, ob die Zahl n eine Primzahl ist. Verwende diese Funktion, um eine Liste der Primzahlen von 1 bis 1000 zu erstellen!

```
FUNCTION prim(n:INTEGER):BOOLEAN;
{true, wenn n Primzahl ist}
var t,tzahl: INTEGER;
{t = möglicher Teiler, tzahl = Teilerzahl}
BEGIN tzahl := 0;
  FOR t := 1 to n DO
    BEGIN IF n MOD t = 0 THEN tzahl := tzahl + 1;
    END;
  IF tzahl = 2 THEN prim := true ELSE prim:= FALSE;
END;
```

Schema der Funktion:



§ 6 Hochauflösende Grafik – Ein Funktionsplotter

Turbo-Pascal unterscheidet zwei Betriebsarten, eine für Textdarstellung mit 80 Spalten und 25 Zeilen sowie eine für hochauflösende Grafik mit einer Standardauflösung von 640 Spalten und 480 Zeilen.
In der Textdarstellung lässt sich ein Zeichen folgendermaßen ausgeben:

```
gotoxy(x1,y1); write('*');
```

Der Ursprung (0/0) des Koordinatensystems liegt dabei links oben

Diese Befehle stehen in der Grafik nicht zur Verfügung. Damit die zahlreichen Prozeduren und Funktionen zur Erstellung von Grafiken zur Verfügung stehen, muss die zweite Zeile eines Programms erweitert werden: USES CRT, GRAPH; Der Grafikbildschirm hat eine Auflösung von 640 x 480 Pixel, wobei der Punkt mit den Koordinaten (0 / 0) links oben liegt.
Von den ca. 50 Grafikprozeduren sind zunächst nur einige wenige wichtig:

1. Grafik einschalten

Dies überträgt man am besten in eine kleine Prozedur:

```
procedure grafikein;
var k, treiber, modus:integer;
begin clrscr;
  treiber:=detect;
  initgraph(treiber,modus,' ');
end;
```

2. Grafik ausschalten

```
procedure grafikaus;
begin repeat until readkey <> '';
    closegraph;
end;
```

3. Hintergrunfarbe wählen

```
setbkcolor(white);
```

4. Zeichenfarbe wählen

```
setcolor(blue);
```

5. Punkt zeichnen

```
putpixel(x,y,farbe);
```

6. Linie zeichnen

```
line(x1,y1,x2,y2); oder lineto(x,y);
```

7. Rechteck zeichnen

```
rectangle(x1,y1,x2,y2);
```

8. Kreis zeichnen

```
circle(x,y,r);
```

9. Text ausgeben

```
procedure
schreibe(x,y:integer;t:string);
begin moveto(x,y);
    settextstyle(1,0,2);
    outtext(t);
end;
```

Weitere Prozeduren behandeln das Zeichnen ausgefüllter Rechtecke mit verschiedenen Füllmustern, die Gestalt der Linien oder die Gestaltung der Textausgabe. Informationen über diese Prozeduren finden sich im Hilfesystem von Turbo-Pascal.

Projekt: Als Anwendung soll ein Programm zur Untersuchung reeller Funktionen geschrieben werden. Das Programm soll modular aufgebaut sein, Teilprobleme sollen in Prozeduren gelöst werden. Das Programm sollte in einer Minimalversion folgendes leisten:

- Wertetabelle erstellen
- Nullstellen suchen
- Einfache Symmetrie erkennen
- Das Schaubild zeichnen
- Das Schaubild der ersten Ableitung zeichnen

Zunächst soll ein Menü angeboten werden, aus dem der Anwender die gewünschte Teilaufgabe durch Tastendruck auswählen kann. Erst bei der Eingabe 0 soll das Programm enden. Das bei Windows-Programmen übliche Rollbalkenmenü ist unter Turbo-Pascal schwierig zu programmieren. Hier erfolgt die Auswahl einfach durch Tippen der passenden Ziffer.

6.1 Das Hauptprogramm

Das Hauptprogramm soll in einer Schleife ein Menü für die einzelnen Programmteile anbieten. Erst bei der Wahl 0 endet das Programm. Die Funktion f wird zu Beginn des Programms ebenfalls definiert.

```
PROGRAM funktionsplotter;
{einfacher Funktionsplotter für Funktionen
Autor: W. Steffen im Info-Kurs 11 (2002), Lebach}
uses crt,graph;

var wahl : CHAR;

FUNCTION f(x:REAL):REAL;
{diese Funktion f wird gezeichnet}
BEGIN f := x*x-2 ;
END;
```

..... {hier folgen die weiteren Prozeduren}

```
BEGIN {hier beginnt das Hauptprogramm}
  REPEAT
    TextColor(yellow);
    TextBackground(blue);
    ClrScr;
    TextColor(lightred);
    WriteLn;
    WriteLn(' F u n k t i o n s p r o g r a m m  1 . 0');
    WriteLn(' ======================================');
    WriteLn;
    WriteLn('  vom Informatik-Kurs 11 - 2002 (Steffen)');
    WriteLn;
    WriteLn('  Die Funktion ist im Programm definiert.');
    WriteLn;
    TextColor(yellow);
    WriteLn('    1 : Wertetabelle von f');
    WriteLn('    2 : einfache Symmetrien von f');
    WriteLn('    3 : Nullstellen von f');
    WriteLn('    4 : Schaubild von f');
    WriteLn('    5 : Schaubild von f mit 1. Ableitung');
    WriteLn;
    TextColor(lightred);
    WriteLn('    0 : E N D E ');
    WriteLn;
    WriteLn('    Bitte wählen!');
    TextColor(yellow);
    wahl := ReadKey;
    IF wahl = '1' THEN wertetabelle;
    IF wahl = '2' THEN symmetrie;
    IF wahl = '3' THEN symmetrie;
    IF wahl = '4' THEN zeichne_schaubild;
    IF wahl = '5' THEN zeichne_ableitung;
  until wahl = '0';
END.
```

6.2 Wertetabelle der Funktion

Zunächst wird der Bereich sowie die Schrittweite der Tabelle eingelesen. Um unsinnige Eingaben zu vermeiden, wird zum Einlesen eine Funktion verwendet, die nur Werte innerhalb vorgegebener Grenzen akzeptiert.

```
FUNCTION eingabe(info:STRING; von,bis:REAL):REAL;
var hilf: REAL;
BEGIN REPEAT
  Write('     ',info,' ');
  ReadLn(hilf);
until (hilf >= von) AND (hilf <= bis);
  eingabe := hilf;
END;
```

In der Prozedur **wertetabelle** erfolgt die Ausgabe der Wertetabelle innerhalb einer repeat-until-Schleife. Damit die Tabelle auf dem Bildschirm sichtbar bleibt, steht am Ende eine Warteschleife, die auf einen beliebigen Tastendruck wartet.

```
PROCEDURE wertetabelle;
{erstellt eine Wertetabelle}
var a, b, dx, x : REAL;
BEGIN ClrScr;
  WriteLn;
  WriteLn('     W E R T E T A B E L L E    V O N   F');
  WriteLn('     ==================================');
  WriteLn;
  a:= eingabe('linker Startwert für x (z.B. -5) : ',
  -100,100);
  b:= eingabe('rechter Startwert für x (z.B. 5) : ',
  -100,100);
  dx := eingabe('Schrittweite für x (z.B. 0.5)   :
  ',0.01,10);
  WriteLn;
  x := a;
  WriteLn('         x         f(x)');
  WriteLn('    ----------------------');
  REPEAT
    WriteLn(x:12:2, f(x):12:4);
    x := x+dx;
  until x > b;
  REPEAT until ReadKey <>'';
END;
```

6.3 Untersuchung auf Symmetrie

Einfache Symmetrien von Funktionen sind

- symmetrisch zur y-Achse, wenn f(-x) = f(x) für alle x
- punktsymmetrisch zum Ursprung, wenn f(-x) = -f(x) für alle x

Eine Möglichkeit zum Erkennen solcher Symmetrien besteht im Vergleich der Funktionswerte an mindestens zwei Paaren von x-Werten, hier –1 und –2 und 2. Damit werden mit hoher Wahrscheinlichkeit (!) die beiden einfachen Symmetrien innerhalb einer if-Anweisung erkannt.

```
PROCEDURE symmetrie;
{prüft die auf Symmetrie}
BEGIN ClrScr;
  WriteLn;
  WriteLn('     S Y M M E T R I E    V O N   F');
  WriteLn('     ============================');
  WriteLn;
  IF (f(-1) = f(1)) AND (f(-2) = f(2)) THEN
   WriteLn('   f ist symmetrisch zur y-Achse.')
  ELSE IF (f(-1) = -f(1)) AND (f(-2) =-f(2)) THEN
   WriteLn('  f ist punktsymmetrisch zum Ursprung.')
  ELSE WriteLn('     f hat keine einfache Symmetrie.');
  REPEAT until ReadKey <>'';
END;
```

6.4 Nullstellen der Funktion

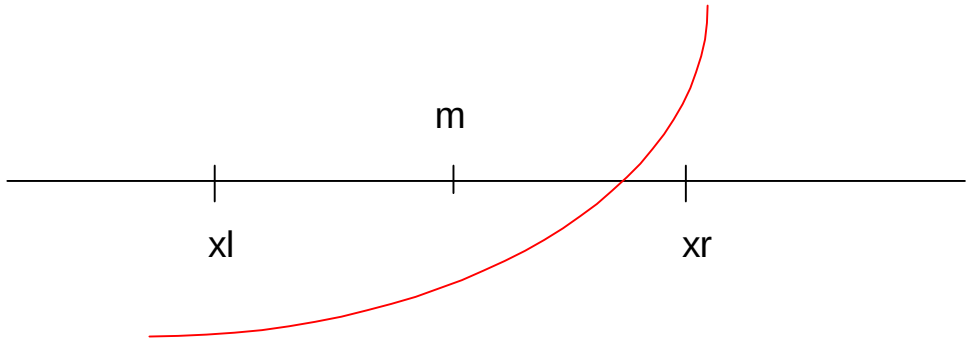
Nullstellen sind x-Werte, an denen der Funktionswert 0 ist. Man unterscheidet Nullstellen **mit** und **ohne** Vorzeichenwechsel. Besonders einfach sind Nullstellen mit Vorzeichenwechsel zu finden. Man zerlegt ein Suchintervall in viele Teilintervalle und testet, ob die Funktion ihr Vorzeichen ändert.
Ist ein solches Intervall gefunden, so werden die Grenzen an eine Funktion **intervallhalbierung** übergeben, in der die Nullstelle weiter eingegrenzt wird.

```
PROCEDURE nullstelle;
{sucht Nullstellen mit Vorzeichenwechsel}
var a,b,x,dx: REAL;
    nst: REAL;

BEGIN ClrScr;
  WriteLn;
  WriteLn('     Nullstellen im Intervall [a,b]');
  WriteLn('     ============================');
  WriteLn;
  WriteLn('      Eingabe des Suchintervalls:');
```

```
  WriteLn;
  a := eingabe('linke Suchgrenze (z.B. -8) : ',-100,100);
  b := eingabe('rechte Suchgrenze (z.B.  8) : ',-90,100);
  WriteLn;
  x := a;
  dx := (b - a) / 99;
  TextColor(lightgreen);
  REPEAT
    IF f(x) * f(x + dx) <= 0 THEN
      BEGIN nst := intervallhalbierung(x , x + dx);
            if abs(f(nst))<1 then WriteLn(nst:15:6);
      END;
    x := x + dx;
  until x >= b;
  TextColor(yellow);
  WriteLn;
  WriteLn('     Fertig!');
  REPEAT until ReadKey <> '';
END; {nullstelle}
```

Vor diese Prozedur muss noch die Funktion intervallhalbierung definiert werden.



Diese geht davon aus, dass die Funktion im Intervall [xl , xr] ihr Vorzeichen ändert. Nun wird der Funktionswert f(m) in der Mitte m des Intervalls berechnet und mit dem Funktionswert am linken Rand xl verglichen. Ändert sich das Vorzeichen im linken Teilintervall, so wird dieses weiterverwendet, sonst das rechte Teilintervall. Dies wird so lange wiederholt, bis die Länge des Intervalls kleiner als eine vorgegeben Schranke ist, hier 0.0000001.

Hinweis: Damit bei Funktionen mit einer unendlichen Sprungstelle wie beispielsweise 0 bei f(x) = 1/x, erfolgt noch eine Prüfung, ob der Funktionswert bei der ermittelten Nullstelle nicht zu groß ist.

```
if abs(f(nst)) < 1 then WriteLn(nst:15:6);
```

```
FUNCTION intervallhalbierung(xl,xr:REAL):REAL;
{grenzt Nullstelle ein}
var m:REAL;
BEGIN REPEAT
  m := (xl+xr)/2;
  IF f(m) * f(xl) <= 0 THEN xr := m ELSE xl := m;
  until (xr - xl) < 0.0000001;
  intervallhalbierung := m;
END; {intervallhalbierung}
```

6.5 Schaubild der Funktion

Bevor mit dem eigentlichen Zeichnen begonnen wird, werden einige Prozeduren zum Ein- und Ausschalten der Grafik sowie zum Beschriften von Grafiken definiert.

```
PROCEDURE grafikein;
{schaltet die hochauflösende Grafik ein}
var k, treiber, modus:INTEGER;
BEGIN ClrScr;
  treiber:=detect;
  InitGraph(treiber,modus,'');
END;

PROCEDURE grafikaus;
{schaltet in den Textmodus zurück}
BEGIN REPEAT until ReadKey <> '';
  CloseGraph;
END;

PROCEDURE schreibe(x,y:INTEGER; t:STRING);
{schreibt einen Text in den Grafikbildschirm}
BEGIN MoveTo(x,y);
  SetTextStyle(1,0,2);
  OutText(t);
END;
```

Der Grafikbildschirm hat bekanntlich eine Auflösung von 640 x 480 Pixel, wobei der Punkt mit den Koordinaten (0 / 0) links oben liegt. Da als Bereich für die x- und y-Achse jeweils das Intervall von -8 bis 8 gewählt wird, hat eine Längeneinheit in x-Richtung 640/16 = 40 Pixel, in y-Richtung sind es 480/16 = 30 Pixel. Da die Prozeduren zum Zeichnen ganze Zahlen erwarten, müssen Werte vom Typ real mit round() gerundet werden.

Das Zeichnen der Koordinatenachsen kann in eine eigene Prozedur ausgelagert werden. Zum Testen reichen zunächst zwei Linien, besser sind aber zusätzliche Teilstriche auf der x- und y-Achse in Einerschritten.

```
PROCEDURE achsen;
{zeichnet das Koordinatensystem}
var h:INTEGER;
BEGIN Rectangle(0,0,639,479);
  {x- und y-Achse}
  Line(0,240,639,240);
  Line(320,0,320,479);
  {jetzt Teilstriche}
  FOR h := 1 to 15 DO
    Line(h*40,230,h*40,250);
  FOR h := 1 to 15 DO
    Line(310,Round(h*30),330,Round(h*30));
END;
```

Auch das Zeichnen des Schaubildes wird in einer Prozedur erledigt. Zunächst wird eine Schrittweite dx:=(16/4000); festgelegt und x auf den linken Startwert -8 gesetzt. Dann wird x jeweils um die Schrittweite erhöht, der Funktionswert y berechnet und der zugehörige Bildpunkt berechnet. Mit ein wenig Dreisatzrechnung werden die reellen Koordinaten x und y in die richtigen Bildschirmkoordinaten xbild und ybild umgerechnet.
Bei der Koordinate ybild für die y-Koordinate ist zu beachten, dass der Ursprung des Koordinatensystems auf dem Bildschirm links oben liegt, die y-Achse also quasi auf dem Kopf steht. Daher wird der passend multiplizierte reelle y-Wert von 240 subtrahiert. Auf dem Bildschirm liegt bei y = 240 gerade die x-Achse.

ybild:=round(240 – 30 * y);

```
PROCEDURE zeichne_schaubild;
{hier wird das Funktionsschaubild gezeichnet}
var x,y,dx:REAL;
    xbild,ybild:INTEGER;
BEGIN grafikein;
  SetBkColor(white);
  SetColor(blue);
  schreibe(40,450,'Schaubild der Funktion');
  achsen;
  x := -8.000001;
  dx:=(16/4000);
```

```
  REPEAT
    y := f(x);
    xbild:=Round((320 + 40 * x));
    ybild:=Round(240 – y * 30);
    IF (ybild < 480) AND (ybild >0) THEN
      PutPixel(xbild,ybild,lightred);
    x := x+dx;
  until x > 8;
  grafikaus;
END;
```

6.6 Schaubild der Ableitung

Aus dem Schaubild der Ableitung lassen sich wichtige Erkenntnisse über die Funktion ablesen. Ist z.B. die Ableitung 0, so hat die Funktion f meistens eine Extremstelle.
Die Ableitung kann näherungsweise als Differenzenquotient berechnet werden.

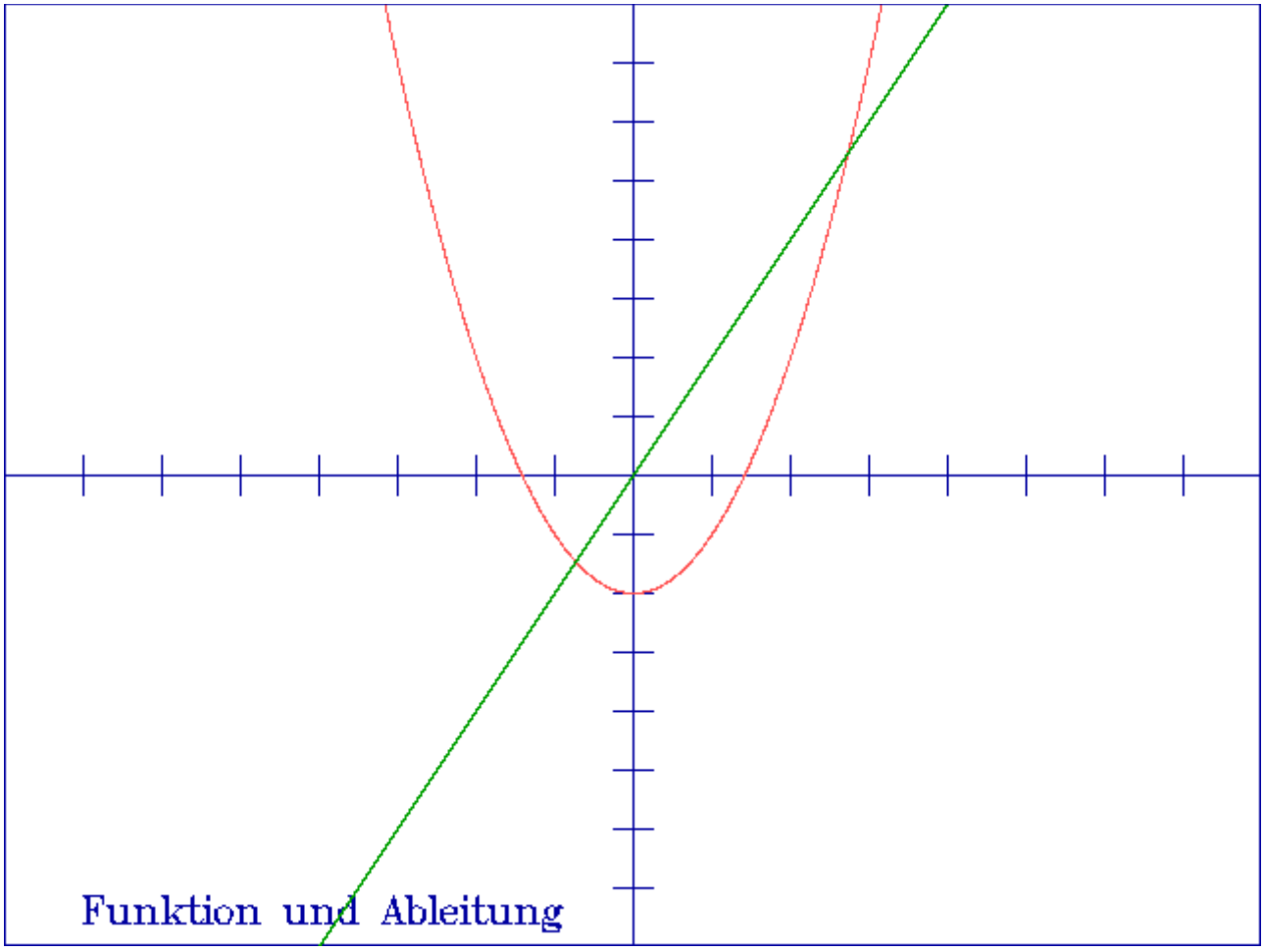
```
abl := (f(x + 0.1) - f(x - 0.1)) / 0.2;
```

Ähnlich wie in der vorherigen Prozedur wird zunächst das Schaubild der Funktion f gezeichnet, anschließend in einer erneuten Schleife das Schaubild der Ableitungs-funktion.

```
PROCEDURE zeichne_ableitung;
{Schaubild von Funktion und Ableitung}
var x,y,dx : REAL;
    abl: REAL;
    xbild,ybild : INTEGER;
BEGIN grafikein;
  SetBkColor(white);
  SetColor(blue);
  schreibe(40,450,'Funktion und Ableitung');
  achsen;
  x := -8.000001;
  dx:=(16/4000);
  REPEAT
    y := f(x);
    xbild:=Round(320 + 40 * x);
    ybild:=Round(240-30 * y);
    IF (ybild < 480) AND (ybild >0) THEN
      PutPixel(xbild,ybild,lightred);
    x := x+dx;
  until x > 8;
```

```
  {jetzt wird die Ableitung gezeichnet}
  x := -8.000001;
  dx:=(16/4000);
  REPEAT
    abl := (f(x + 0.1) - f(x - 0.1)) / 0.2;
    xbild:=Round(320 + 40 * x);
    ybild:=Round(240 – 30 * abl);
    IF (ybild < 480) AND (ybild > 0) THEN
      PutPixel(xbild,ybild,green);
    x := x+dx;
  until x > 8;
  grafikaus;
END;
```

Für die Funktion f mit f(x) = x*x – 2 wird durch unser Programm ein recht ansprechendes Schaubild erzeugt. Die Funktion wird rot, die Ableitung grün gezeichnet.



§ 7 Felder und Dateien

Die bisherigen Probleme waren dadurch gekennzeichnet, dass mit wenigen Daten relativ komplizierte Berechnungen durchgeführt wurden. In vielen Fällen sind dagegen sehr viele Daten vorhanden, mit denen relativ einfach gearbeitet wird: Werte eingeben, auf den Bildschirm schreiben, auf die Festplatte speichern, den größten Wert suchen, sortieren usw.

Beispiele: Monatsstatistik über Niederschläge
Rangliste eines Sportsvereins
Namensliste einer Klasse oder einer ganzen Schule
Notenstatistik einer Klassenarbeit

Für die genannten Beispiele eignet sich der Datentyp Feld bzw. array.
Ein ***Array*** (Feld) ist

- eine Aneinanderreihung von ***Elementen des gleichen Typs***,
- auf welche durch einen ***Index*** zugegriffen wird.

7.1 Vereinbarung eines Feldes

Damit die Feldvariablen später auch als Parameter an Prozeduren und Funktionen übergeben werden können, muss zuerst ein eigener Datentyp vereinbart werden.

```
Type feldtyp = array [ 1..31 ] of integer;
Var niederschlag : feldtyp;
```

Dadurch wird eine Variable niederschlag definiert, die aus 31 ganze Zahlen als Komponenten enthält. Der Zugriff erfolgt über die Elementnummer, den Index.
niederschlag [5] := 12;

Beispiel: Namensliste eines Informatikkurses

Anhand einer einfachen Namensliste sollen einige Aspekte zum Umgang mit Feldern dargestellt werden. Das Feld besteht hier lediglich aus 14 Komponenten, in denen die Namen abgelegt werden.

```
program array_test;
uses crt;

type inhalttyp = string[25];
     feldtyp = array[1..14] of inhalttyp;
var f : feldtyp;
```

Es empfiehlt sich, ähnlich wie im Funktionsprogramm wieder ein Auswahlmenü zu entwerfen, in dem dann die verschiedenen Aktionen ausgewählt werden können.

7.2 Eingeben des Feldes

Typisch für das Arbeiten sind Zählschleifen, also Wiederholungen mit for-to. Die Eingabe des Feldes erfolgt in einer Prozedur. Wichtig ist es, die Feldvariable *f* mit var als Ausgabeparameter zu kennzeichnen.

```
procedure feldeingabe(var f : feldtyp);
var k : integer;
begin clrscr;
      writeln('   Eingabe der Namen');
      writeln;
      for k := 1 to 14 do
      begin write('    Nummer ',k,' : ');
            readln(f[k]);
      end;
end; {feldeingabe}
```

7.3 Ausgeben des Feldes

Auch für die Ausgabe des Feldes wird lediglich eine Zählschleife benötigt. Es ist sinnvoll, die benötigte Zählvariable k lokal und nicht etwa global im Hauptprogramm zu definieren.

```
procedure ausgabe(f:feldtyp);
var k : integer;
begin clrscr;
      writeln('     Ausgabe des Feldes');
      writeln;
      for k := 1 to 14 do
         writeln(k:8,'  ',f[k]);
      repeat until readkey <> '';
end; {feldausgabe}
```

7. 4 Abspeichern des Feldes

Hat man – eventuell mühevoll – alle Elemente eines größeren Feldes eingegeben, so kommt bald der Wunsch auf, diese Daten dauerhaft auf der Festplatte oder einer Diskette zu speichern. Dazu muss in Pascal eine Dateivariable vereinbart werden.

var datei : file of feldtyp;

Die Variable *datei* entspricht einer Datei auf der Festplatte, in der hier nur eine Variable, nämlich die Feldvariable f mit allen Namen, abgespeichert wird.

Die Zuordnung Dateivariable <---> Dateinamen erfolgt durch die Anweisung *assign(datei, dateiname).* "Dateiname" ist dabei ein gültiger Dateiname z.B. "array.dat".

Ist diese Zuordnung erfolgt, so stehen u.a. folgende Prozeduren und Funktionen zu Verfügung.

Rewrite(datei);	öffnet die Datei zum Schreiben
Reset(datei);	öffnet die Datei zum Lesen
Write(datei,variable);	schreibt eine Variable in die Datei
Read(datei,variable);	liest eine Variable aus der Datei
Close(datei);	schließt die Datei

Damit kann die Prozedur zum Abspeichern leicht formuliert werden. Im Beispiel wird auf der Diskette im Laufwerk a: die Datei "array.dat" erzeugt.

```
procedure speichern(f : feldtyp);
var datei : file of feldtyp;
begin assign(datei,'a:\array.dat');
      rewrite(datei);
        write(datei,f);
      close(datei);
      clrscr;
      gotoxy(20,20);
      write('Daten wurden erfolgreich abgespeichert.');
      repeat until readkey <> '';
end; {speichern}
```

7.5 Laden des Feldes

Beim Laden darf man nicht vergessen, die Parameter im Prozedurkopf mit dem Schlüsselwort *var* als Ausgabeparameter zu kennzeichnen. Nur so stehen die Daten auch in den anderen Prozeduren zur Verfügung.

```
procedure laden(var f: feldtyp);
var datei : file of feldtyp;
begin clrscr;
      assign(datei,'a:\array.dat');
      reset(datei);
      read(datei,f);
      gotoxy(20,20);
      write('Daten wurden erfolgreich geladen.');
      close(datei);
      repeat until readkey <> '';
end;{laden}
```

Für Dateivariable gibt es weitere Prozeduren, mit denen beispielsweise eine Datei umbenannt oder gelöscht werden kann. Interessant ist die Funktion

function eof(datei: dateityp):boolean);

Sie prüft, ob das Dateiende beim Lesen schon erreicht ist. Wenn man beim Lesen einer Datei nicht weiß, wie viele Elemente sie enthält, so ist die folgende Anweisung eine sichere Methode.

While not eof(datei) do read(datei, variable);

7.6 Sortieren der Feldelemente

Das Sortieren von Daten nach Größe bzw. wird oft benötigt. Man denke an ein Telefonbuch, in dem man einen Teilnehmer nur dann schnell findet, wenn die Liste alphabetisch sortiert ist. Zum Sortieren gibt es verschiedene Algorithmen. Der einfachste ist das Verfahren "Bubble-Sort-Verfahren", auch "Sortieren durch Austausch" genannt. Dabei wird das Feld von vorne an durchlaufen und jeweils zwei benachbarte Feldelemente verglichen. Kommt das größere Element zuerst, so wird es mit seinem Nachbarn vertauscht.

Zum Veratuschen wird eine Hilfsvariable benötigt. Nach den beiden Anweisungen `f[k] := f[k+1] ; f[k+1] := f[k] ;` stünde in beiden Feldkomponenten der gleiche Name. Daher muss zunächst der Inhalt von f[k] in eine Hilfsvariable "gerettet" werden.

Offenbar wandert bei diesem Verfahren das größte Element nach oben, ähnlich wie die Gasblasen in Getränken, daher der Name "Bubble-Sort".

Zwar steht nach dem ersten Durchgang das größte Element oben, das kleinste Element allerdings noch nicht im Feldelement f[1]. Das Durchlaufen des Feldes muss also mehrmals wiederholt werden, bis keine Vertauschung mehr erforderlich war. Dies wird durch eine bool'sche Variable ok kontrolliert.

```
procedure sortieren(var f:feldtyp);
var k: integer;
    hilf : inhalttyp;
    ok : boolean;
begin clrscr;
      repeat
      ok := true;
      for k := 1 to 13 do
        if f[k] > f[k+1] then
        begin hilf := f[k];
              f[k] := f[k+1];
              f[k+1] := hilf;
              ok := false;
      end;
      until ok = true;
      gotoxy(20,20);
      write('Feld wurde erfolgreich sortiert.');
      repeat until readkey <> '';
end; {sortieren}
```

Für kleine Felder mit wenigen Elementen ist dieser Sortieralgorithmus gut geeignet. Bei sehr vielen zu sortierenden Daten steigt die Rechenzeit dagegen stark an, so dass dann schnellere Sortierverfahren erforderlich sind.

Aufgaben: 1. Definiere ein Feld mit 31 Elementen zur Aufnahme von täglichen Niederschlagsmengen! Schreibe Prozeduren zur Ein- und Ausgabe sowie zur Ermittlung des größten und kleinsten Feldelementes!

2. Schreibe ein Programm, das eine Pascaldatei (Datentyp *file of char*) öffnet und zeichenweise auf dem Bildschirm ausgibt! Teste auch die Funktion *upcase(z)*, die Kleinbuchstaben in Großbuchstaben umwandelt!