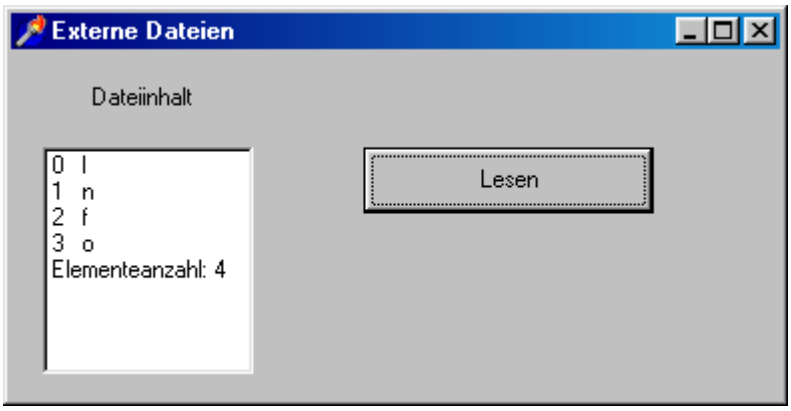



```
var  Form1: TForm1;
     Dat: File of CHAR;

procedure TForm1.Start(Sender: TObject);
VAR ch: CHAR;
begin
 AssignFile(Dat,'Buchstabendatei');
 Rewrite(Dat);
 ch := 'I'; Write(Dat,ch);
 ch := 'n'; Write(Dat,ch);
 ch := 'f'; Write(Dat,ch);
 ch := 'o'; Write(Dat,ch);
 CloseFile(Dat)
end;

procedure TForm1.LesenClick(Sender: TObject);
VAR c: CHAR;
    i:INTEGER;
begin
  Listbox1.Clear;
  Reset(Dat);
  WHILE Not EOF(Dat) DO BEGIN
    i := FilePos(Dat);
    Read(Dat, c);
    Listbox1.Items.Add(IntToStr(i)+'   '+c)
  END;
  Listbox1.Items.Add('ElAnzahl: '+IntToStr(FileSize(Dat)));
  CloseFile(Dat)
end;

end.
```

Datei-Befehle

procedure **AssignFile**(**var** F: Dateityp; FileName: string);
Diese Prozedur ordnet den Namen einer externen Datei einer Dateivariablen zu.
Beispiel: VAR dat: **File of Integer**;
....
AssignFile(dat, 'Primzahldatei')

procedure **Rewrite**(**Var** F: Dateityp);
Diese Prozedur erstellt **eine neue, leere Datei** und öffnet sie zum Schreiben (es gibt ja noch nichts zu lesen! Allerdings kann in einer so geöffneten Datei durchaus auch schon gelesen werden, natürlich erst nachdem man ein paar Daten geschrieben hat und z.B. mit *Seek(F, 0)* auf den Anfang dieser so geöffneten Datei gesprungen ist). Ist schon eine gleichnamige externe Datei vorhanden, wird sie gelöscht und an ihrer Stelle die neue Datei angelegt.
Ist F bereits offen, wird sie zuerst geschlossen und dann erneut erstellt. Die Datei wird also auf jeden Fall entweder neu erstellt oder überschrieben! Nach dem Erstellen wird der Dateizeiger an den Anfang der leeren Datei gesetzt, die aktuelle Dateiposition *FilePos(F)* ist also 0.

procedure **Write**(**Var** F: Dateityp; **Var** d: Datentyp);
Diese Prozedur schreibt Daten in eine Datei. Nach jedem Schreiben einer **Variablen** wird die aktuelle Dateiposition um 1 erhöht. Wichtig: der zu schreibende Datensatz d muß eine Variable sein, kann also nicht einfach ein Wert sein wie z.B. 17 oder i*i .

procedure **CloseFile**(**Var** F: Dateityp);
Diese Prozedur geht zunächst bis ans Ende der Datei und schließt sie dann.

procedure **Reset**(**Var** F: Dateityp);
Diese Prozedur öffnet eine **bereits vorhandene** Datei F zum Lesen und/oder zum Schreiben. Ist F bereits offen, wird sie zuerst geschlossen und dann erneut geöffnet. Nach dem Öffnen wird der Dateizeiger an den Anfang der Datei gesetzt, die aktuelle Dateiposition ist 0.

function **EOF**(**var** F: Dateityp): Boolean;
Diese Funktion prüft, ob das Dateiende erreicht ist. EOF(F) gibt *True* zurück, wenn sich die aktuelle Dateiposition hinter dem letzten Zeichen der Datei befindet, oder wenn die Datei leer ist. Andernfalls wird *False* zurückgegeben.

procedure **Read**(**Var** F: Dateityp; **Var** x: Datentyp);
Diese Prozedur liest Daten aus einer Datei. Nach jeder Read-Anweisung wird die aktuelle Dateiposition um 1 erhöht. Wichtig: der zu lesende Datensatz x muß eine Variable sein.

function **FilePos**(**Var** F: Dateityp): LongInt;
Diese Funktion gibt die aktuelle Dateiposition zurück. FilePos kann nicht für **Text**dateien verwendet werden. Für den Dateianfang gilt: FilePos(F) = 0

procedure **Seek**(**var** F; N: Longint);
Der Positionszeiger der Datei F wird auf die Komponente mit der Nummer N gesetzt. Die erste Komponente in einer Datei hat immer die Nummer 0.

function **FileSize**(**var** F): Integer;
Diese Funktion gibt die Anzahl der Elemente einer Datei zurück. Die Funktion kann nur für geöffnete Dateien verwendet werden. FileSize kann nicht für Textdateien verwendet werden.

function **FileExists**(FileName: **string**): Boolean;
Diese Funktion gibt True zurück, wenn die im Parameter FileName angegebene Datei vorhanden ist. Existiert die Datei nicht, wird False zurückgegeben.

function **RenameFile**(Const OldName, NewName: STRING): BOOLEAN;
Diese Funktion versucht, den Dateinamen zu ändern. Falls es nicht geklappt hat (weil z.B. die Datei schreibgeschützt war oder der neue Name schon existiert), wird *False* zurückgeliefert. Falls es geklappt hat, wird *True* zurückgeliefert.

function **DeleteFile**(Const DateiName: STRING): BOOLEAN;
Diese Funktion versucht, die angegebene Datei zu löschen. Falls es nicht geklappt hat (weil z.B. der Dateiname nicht existiert oder die Datei schreibgeschützt war), wird *False* zurückgeliefert, ansonsten *True*.

Aufgabe

Erstelle ein einziges Programm, welches nach und nach alle folgenden Aufgaben löst (für jede Aufgabe einen eigenen Button!):

a) Erzeuge eine Datei namens *Quadrat*, welche die ersten 10 Quadratzahlen enthält und eine Datei namens *Kubik*, welche die ersten 10 Kubikzahlen enthält!

b) Die Elemente der Datei *Quadrat* werden in der Listbox1 angezeigt.
Die Elemente der Datei *Kubik* werden in der Listbox2 angezeigt.

c) In einem Editfeld wird eine Zahl **n** eingegeben. Es soll überprüft werden, ob diese Zahl **n** in der Datei *Quadrat* enthalten ist oder nicht. Das Ergebnis wird in Form einer Showmessage ausgegeben.

d) In einem Editfeld wird eine Zahl **n** eingegeben. Diese Zahl **n** wird an die Datei *Quadrat* angehängt. Anschließend wird diese Datei in der Listbox1 ausgegeben.

e) In einem Editfeld wird eine Nummer **n** eingegeben. Die entsprechende **n**-te Zahl in der Datei *Quadrat* wird durch die Zahl 999 ersetzt. Anschließend wird zur Kontrolle die neue Datei *Quadrat* in der Listbox1 ausgegeben

f) In einem Editfeld wird eine Zahl **n** eingegeben. Falls diese Zahl in der Datei *Quadrat* enthalten sein sollte, so wird sie durch die Zahl 999 ersetzt. Anschließend wird zur Kontrolle die neue Datei *Quadrat* in der Listbox1 ausgegeben

g) In einem Editfeld wird eine Nummer **n** eingegeben. Die entsprechende **n**-te Zahl in der Datei *Quadrat* wird gelöscht.
Einfache Lösung: Alle Zahlen der Datei *Quadrat* mit Ausnahme der **n**-ten werden in eine zweite Datei *Quadrat2* geschrieben. Anschließend wird die erste Datei *Quadrat* gelöscht und die zweite Datei *Quadrat2* umbenannt in *Quadrat*.

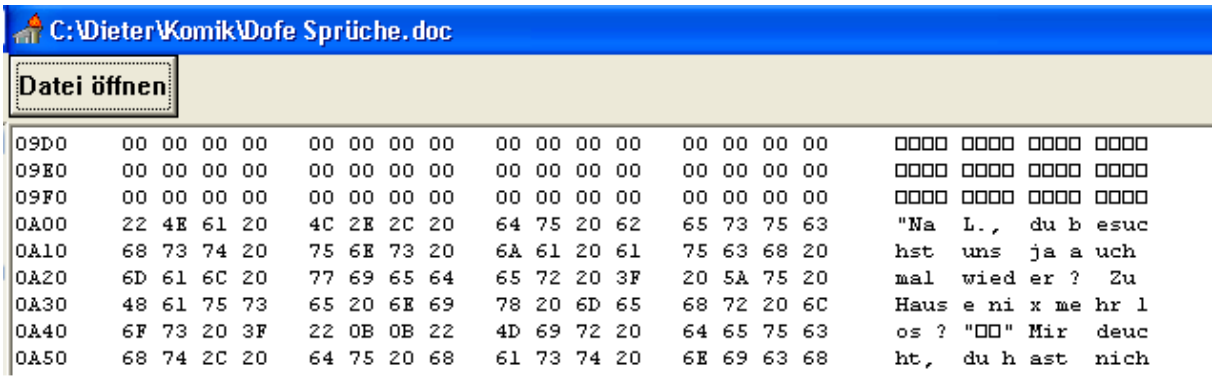
h) Die beiden Dateien *Quadrat* und *Kubik* werden hintereinander gehängt zu einer dritten Datei namens *Zahlen*. Diese wird in der Listbox3 dargestellt.

i) Die beiden Dateien *Quadrat* und *Kubik* werden (der Größe nach sortiert) zusammengemischt zu einer vierten Datei namens *ZahlenSortiert*. Diese wird in der Listbox3 dargestellt.


```
var  Main: TMain;
     f: File Of INTEGER;

procedure TMain.Ausgabe;
VAR zahl: INTEGER;
BEGIN
 ListBox.Clear;
 Reset(f);
 WHILE NOT EOF(f) DO BEGIN
   read(f, zahl);
   ListBox.Items.Add(IntToStr(zahl));
 END;
 CloseFile(f)
END;

procedure TMain.BtErzeugenClick(Sender: TObject);
VAR i, zahl: INTEGER;
begin
  AssignFile(f, 'Zahlendatei');  Rewrite(f);
  FOR i := 1 TO 10 DO BEGIN
    zahl := i*i;  write(f, zahl)
  END;
  CloseFile(f);  Ausgabe
end;

procedure TMain.BtErsetzenClick(Sender: TObject);
VAR zahl: INTEGER;
begin
  Reset(f);
  Seek(f,4);
  zahl := 1000;  write(f, zahl);
  CloseFile(f);  Ausgabe
end;

procedure TMain.BtAnhaengenClick(Sender: TObject);
VAR zahl: INTEGER;
begin
  Reset(f);
  Seek(f,FileSize(f));
  zahl := 1000;
  write(f, zahl);
  CloseFile(f);  Ausgabe
end;

end.
```

HEX-DUMP

Mit dem folgenden Programm kann man den Inhalt beliebiger Dateien auflisten. Wichtig: Die ausgesuchte Datei darf nicht schreibgeschützt sein, obwohl nur gelesen wird.
Angenommen, man interessiert sich für den Inhalt irgendeiner Datei namens *„Willi.abc“*. Dann erzeugt man im Programm eine Datei f vom Typ Byte und assoziiert sie mit dem gewünschten Dateinamen *„Willi.abc“*. Normalerweise enthält die Datei *Willi* etwas völlig anderes als Bytes. Wichtig ist nur, dass alles, was gespeichert wird, eben in Bytes gespeichert wird und als solche Datei gelesen werden kann (selbst wenn es sich um Musikdateien oder um Bilder handeln sollte).
In der folgenden Abbildung werden in jeder Zeile 16 Bytes der Datei dargestellt. Falls es sich um ASCII-Codes von Buchstaben, Ziffern oder Satzzeichen handeln sollte, werden diese am rechten Rand dargestellt. Ansonsten sieht man am rechten Rand ein rechteckiges Kästchen.
In der linken Spalte ist hexadezimal jeweils die Nummer des ersten Elementes dieser Zeile angegeben.




```
unit Hexfile; ……….……
type  TMain = class(TForm)
        Oeffnen: TButton;
        OpenDialog1: TOpenDialog;
        Memo1: TMemo;
        procedure OeffnenClick(Sender: TObject);
      private
        f: File Of Byte;
      end;

var  Main: TMain;
```

```
implementation…….

procedure TMain.OeffnenClick(Sender: TObject);
VAR i, j, zeile: INTEGER;
    zahl: BYTE;
    s1, s2: STRING;

begin
  Memo1.Clear;
  zeile := 0;

  With OpenDialog1 DO Begin
    Title := 'Datei laden';
    Filter := 'alle Dateien *.*|*.*';
    InitialDir := 'C:\Dieter'
  End;

  If OpenDialog1.Execute THEN Begin
    Caption := OpenDialog1.Filename;
    AssignFile(f, OpenDialog1.Filename);
    reset(f);

    WHILE NOT EOF(f) DO Begin
      i := 1;
      s1 := '';
      s2 := '';
      WHILE NOT EOF(f) AND (i<=16) DO BEGIN
        read(f, zahl);
        s1 := s1 + IntToHex(zahl, 2) + ' ';
        If zahl > 31 THEN s2 := s2 + chr(zahl)
        ELSE s2 := s2 + chr(1);
        i := i+1
      END;

      If i <= 16 THEN FOR j := i TO 16 DO BEGIN
         s1 := s1 + '   ';
         s2 := s2 + ' '
      End;
```

```
      INSERT('  ', s1, 12);
      INSERT('  ', s1, 27);
      INSERT('  ', s1, 41);
      INSERT(' ', s2, 5);
      INSERT(' ', s2, 10);
      INSERT(' ', s2, 15);
      Memo1.Lines.Add(IntToHex(zeile, 4)+ '    '+ s1
+ '    ' + s2);
      zeile := zeile + 16
    END;
    CloseFile(f);
  END; {if Opendialog1 }

end;
end.
```

Dateiaufgaben

1. In dieser Aufgabe soll untersucht werden, wie das Delphi-System Zahlen und Zeichen extern speichert.
   a) Erstelle eine Zahlendatei namens *GanzZahlen.int* ! Diese soll nur die drei INTEGER-Zahlen 1, 5 und 7 enthalten.
   b) Erstelle eine Zahlendatei namens *RealZahlen.real* ! Diese soll nur die drei REAL-Zahlen 1, 5 und 7 enthalten.
   c) Erstelle eine Zeichendatei namens *Zeichen.txt* ! Diese soll nur die drei Zeichen (Typ CHAR) ‚1', ‚5' und ‚7' enthalten.

   Öffne jeweils eine dieser gerade erstellten drei Dateien mit dem HEX-Dump-Programm ! Kannst du erkennen, wie die entsprechenden Inhalte gespeichert wurden?

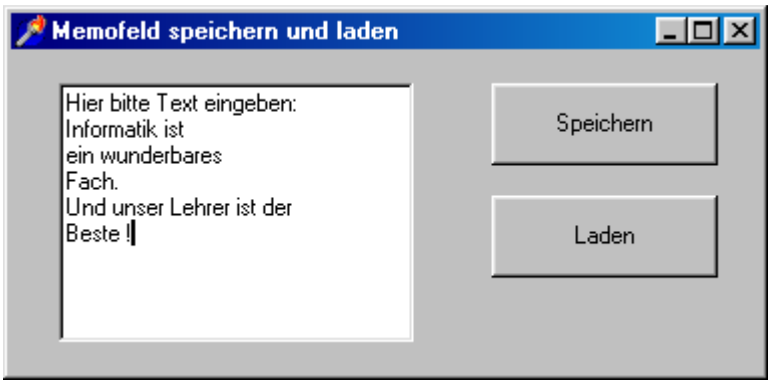
2. Schreibe ein Programm, welches eine der drei Dateien aus der Aufgabe 1 löscht. Wie reagiert das Windows- bzw. Linux-System darauf? Kommen Nachfragen wie „*Möchten Sie wirklich löschen*"?

3. Schreibe ein Delphi-Programm, welches eine der drei Dateien aus der Aufgabe 1 umbenennt. Wie reagiert das Windows-bzw. Linux-System?

4. Erstelle ein kleines sog. Demo-Programm, welches u.a. bei jedem Aufruf des Programms einen Zähler erhöht. Dieser Zähler muss deshalb extern gespeichert werden. Das Demo-Programm darf nur dreimal laufen. Danach muss eine entsprechende Mitteilung an den Benutzer erfolgen und das Programm geschlossen werden (dazu reicht der Befehl ***Close***).

5. Erstelle eine INTEGER-Datei, welche 20 ganzzahlige Zufallszahlen zwischen 1 und 6 enthält. Schreibe anschließend eine Prozedur, welche in dieser Datei jedes Auftreten der Zahl 5 löscht! Das Funktionieren deines Programms sollte mit einer Listbox (für die Dateiausgabe) kontrolliert werden können.

6. Erstelle eine Codezahl-Datei, welche nur eine einzige, fünfstellige INTEGER-Zahl enthält. Schreibe anschließend ein Programm, welches vom Benutzer die Eingabe dieser geheimen Code-Zahl verlangt. Bei richtiger Eingabe erscheint ein Willkommensgruß, bei falscher Eingabe wird das Programm nach entsprechender Mitteilung sofort geschlossen.




```
var  Form1: TForm1;
     F: Text;

procedure TForm1.Start(Sender: TObject);
begin
 Memo1.Clear;
 Memo1.lines.Add('Hier bitte Text eingeben:  ');
 AssignFile(F,'Textdatei')
end;

procedure TForm1.SpeichernClick(Sender: TObject);
VAR i: INTEGER;
begin
  Memo1.Lines.Delete(0);
  Rewrite(F);
  FOR i := 0 TO Memo1.Lines.Count - 1 DO
    writeLN(F,Memo1.Lines[i]);
  //alternativ:  writeLN(F, Memo1.Text)
  CloseFile(F)
end;

procedure TForm1.LadenClick(Sender: TObject);
Var Zeile: STRING;
begin
 Memo1.Clear;
 Reset(F);
 WHILE NOT EOF(F) DO BEGIN
   Readln(F, Zeile);
   Memo1.lines.Add(Zeile)
 END;
 CloseFile(F)
end;

end.
```

Textdateien

VAR F: Text; // nicht möglich als **lokale** Variable !

In Textdateien werden die Texte zeilenweise (beendet mit EOL) gespeichert. Die Zeilen sind üblicherweise unterschiedlich lang. Deshalb ist auch ein direkter Zugriff auf eine bestimmte Zeile nicht möglich (die Seek-Prozedur gibt es hier nicht).

procedure **Write**(**Var** F: Text; **Var** d: STRING);
procedure **WriteLN**(**Var** F: Text; **Var** d: STRING);
Diese Prozeduren schreiben Text in eine Datei. Wichtig: auch der zu schreibende String d muß eine Variable sein.

procedure **Writeln**(**Var** F: Text) schreibt nur ein Zeilenendezeichen in eine **Text**datei.

procedure **Read**(var F: Text; Var d: String);
Read liest alle Zeichen bis zum nächsten Zeilenendezeichen EOL (ausschließlich) oder bis EOF(F) den Wert True hat. Da das Zeilenendezeichen nicht gelesen wird, springt Read nicht an den Anfang der nächsten Zeile. Ist der Ergebnis-String länger als die maximale Länge der String-Variablen, wird er abgeschnitten.

Nach dem ersten Read-Aufruf liest jede nachfolgende Read-Operation das Zeilenendezeichen und gibt einen String der Länge Null zurück.

procedure **Readln**(var F: Text; Var d: String);
Readln liest eine Textzeile aus der angegebenen Datei und setzt den Dateizeiger in die nächste Zeile.

procedure **Append**(**var** F: Text);
Diese Prozedur öffnet die Datei und setzt die aktuelle Dateiposition auf das Dateiende. Falls die Datei schon geöffnet sein sollte, wird sie zuerst geschlossen und dann erneut geöffnet. Append soll nur für Textdateien benutzt werden, funktioniert aber anscheinend (in Delphi 4) auch für andere Dateitypen.

Textdateiaufgaben

Für alle folgenden Aufgaben ist es hilfreich und sinnvoll, wenn man das Ergebnis seiner Programmierung mit Hilfe von Memo-Feldern auf dem Formblatt kontrollieren kann. Auch die Voraussetzungen für einige Aufgaben, dass nämlich schon entsprechende Textdateien existieren, könnte man leicht mithilfe eines Memofeldes oder mit einem Texteditor schaffen. Die eigentliche Aufgabenstellung soll jedoch nicht mit den Fähigkeiten von Memofeldern gelöst werden!

1. Eine „bereits existierende“ Textdatei namens *Passwort.txt* enthält nur ein einziges Wort (das Paßwort). Schreibe ein kleines Programm, welches u.a. dieses Paßwort aus der Datei einliest. Der Benutzer wird nach diesem Paßwort gefragt. Bei falscher Antwort wird das Programm sofort (nach entsprechender Mitteilung mit Showmessage) geschlossen (dazu reicht der Befehl ***Close***). Bei richtiger Antwort erhält der Benutzer die Möglichkeit, dieses Passwort zu ändern. Die Änderung wird natürlich gespeichert.

2. Eine „bereits existierende“ Textdatei, die mehrere Zeilen enthält, soll so in eine zweite Textdatei kopiert werden, dass alle Zeilen nun vorangestellte Zeilennummern (beginnend mit der Nummer 1) erhalten.

3. Zwei „bereits existierende“ Textdateien sollen hintereinander gehängt werden und so eine dritte Textdatei bilden.

4. Eine „bereits existierende“ Textdatei enthält in jeder Zeile einen einzigen Vornamen. Schreibe ein Programm, welches den Namen *„Klaus“* in dieser Datei sucht. Anschließend soll
   a) eine entsprechende Meldung ausgegeben werden.
   b) der Name *„Klaus“* gelöscht werden.
   Hinweis: Achte darauf, ob sich *„Klaus“* am Anfang, in der Mitte oder am Ende der Datei befindet!

5. Untersuche mithilfe des früher geschriebenen Programms *„Hex-Dump“*, wie eine Textdatei gespeichert wird. Achte insbesondere auf das Zeichen EOL!