Einführung in die Informatik, Algorithmen und Datenstrukturen

Teil 1 - Thema 6

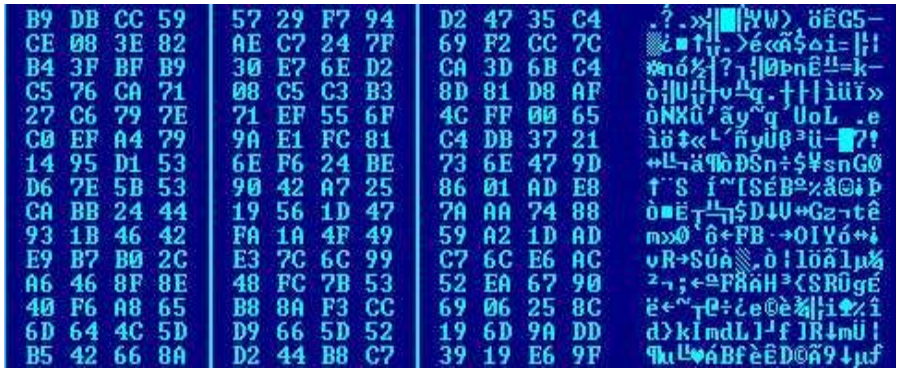
Datenverwaltung mit Object-Pascal

Daten - Definitionen

Foto?

Text?

Musik?



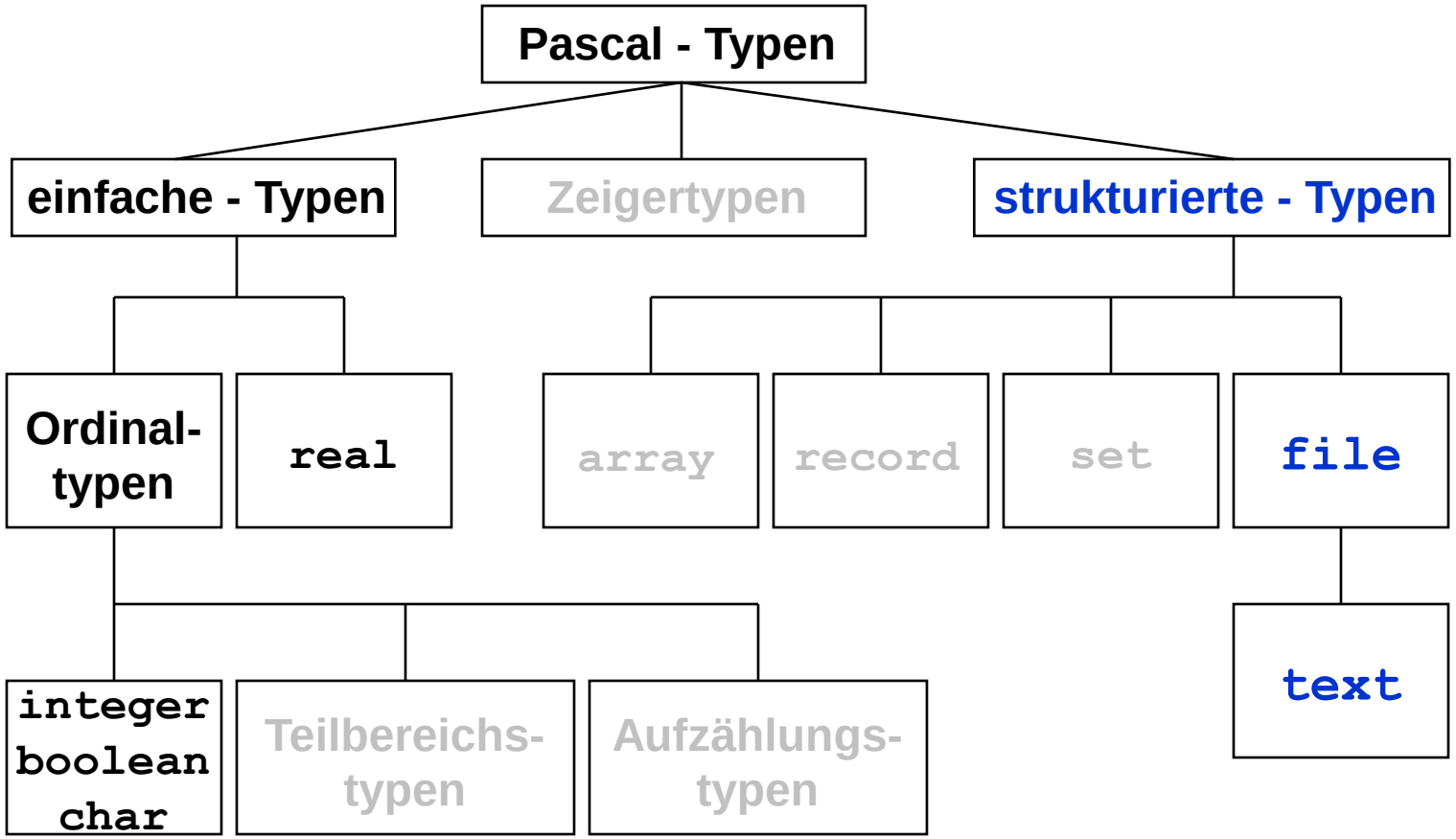
Video?

Zeichnung?

Programm?

Daten, urspr. svw. geschichtl. Zeitangaben; heute allg. Bez. für die Zahlenwerte der Merkmalsgrößen von physikal.-techn. Objekten *(Kenndaten),* Ereignissen, Prozessen und Abläufen (z. B. *Betriebsdaten* bei techn. Vorgängen und Geräten, *Bahndaten* der Bewegungen von Raumflugkörpern). In der Informatik sind D. durch Zeichenfolgen *(digitale D.)* oder kontinuierl. Funktionen *(analoge D.)* dargestellte Informationen oder Sachverhalte, die maschinell bearbeitet werden können.

Typ-Terminologie in Pascal



Textdateien

Textdateien

Eine Textdatei ist eine Folge von Zeilen, wobei eine Zeile aus einer beliebigen Anzahl von Zeichen besteht, auf die ein Zeilenvorschub (CR - LF: Hexadezimal: 0D 0A) folgt. Da die Zeilen in der Regel unterschiedlich lang sind, ist eine berechnete Positionierung nicht möglich. Textdateien werden vom Dateianfang elementweise in Richtung Dateiende beschrieben.

Die Datei-Variable wird als

VAR dateivariable : textfile deklariert.

Neben den Standard-Ein- / Ausgabeprozeduren READ und WRITE sind für Textdateien zusätzlich die Standardprozeduren READLN und WRITELN verfügbar. Bei Textdateien können entweder nur Schreib- oder nur Lesezugriffe ausgeführt werden.

Typisierte Dateien

Typisierte Dateien

Typisierte Dateien bestehen aus einer Folge von Komponenten des gleichen Typs. Die Datei-Variable wird als

VAR dateivariable : FILE OF komponententyp

deklariert.

Als Komponententyp ist jede beliebige Datenstruktur zugelassen. Da jede Komponente einer typisierten Datei denselben Platzbedarf hat, sind berechnete Positionierungen möglich. Alle Operationen haben den Komponententyp als Grundlage. Bei typisierten Dateien können nach der Eröffnung Schreib- und Lesezugriffe parallel durchgeführt werden.

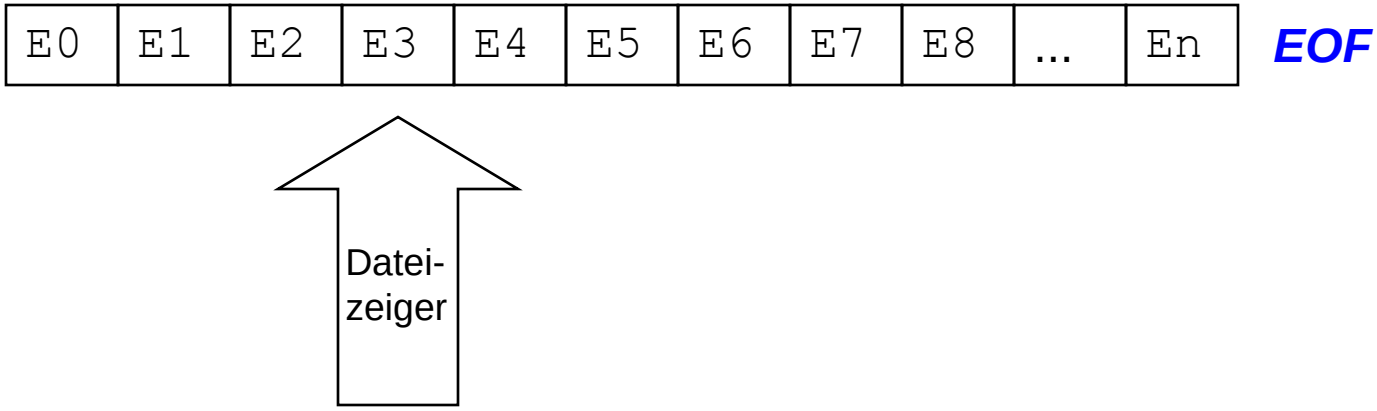
Dateiformen

Physikalisch werden **alle Dateien auf dem Datenträger in der gleichen Form gespeichert** - als eine Folge von einzelnen Bytes. In der Datei sind keine Informationen über den Typ der Elemente enthalten. Daher ist es möglich, eine gespeicherte Datei als Datei eines anderen Typs wieder zu eröffnen und zu bearbeiten.

Diese Dateiform entspricht Dateien mit sequentiellem Zugriff (**sequential access files**). Bei dieser Dateiform werden alle Daten hintereinander gespeichert. Es kann nur in dieser Reihenfolge auf sie zugegriffen werden. Die Aufzeichnung erfolgt analog der Aufzeichnungsform auf Magnetbänder. Jedoch erlaubt die Verwendung von Disketten- und Festplattenlaufwerken eine schnelle Berechnung der Elemente und den Zugriff, ohne alle vorhergehenden Elemente zu lesen.

Aufbau einer sequentiellen Datei

Alle Datenelemente werden in der Reihenfolge ihrer Eingabe gespeichert. Dadurch hat jedes Element automatisch eine Nummer, die Platznummer. Das erste Element hat immer die Platznummer 0. Das Ende der Datei wird durch ein Element mit der Bezeichnung EOF (end of file) gekennzeichnet. Eine leere Datei besteht mindestens aus dem (virtuellen) Element EOF. Die Datenelemente der Datei haben alle den gleichen Typ. Der Typ wird im Deklarationsteil als Dateideklaration vereinbart.



Aufbau einer sequentiellen Datei

Der **Dateizeiger (Filepointer)** stellt die Verbindung zum Dateielement her. Durch die Prozedur READ wird ein Element aus der Datei gelesen. Durch die Prozedur WRITE wird ein Element in die Datei geschrieben. Es besteht die Möglichkeit abzufragen, ob das Ende einer Datei erreicht ist - der Dateizeiger auf EOF weist. Jede **Schreib- / Leseoperation** positioniert den **Dateizeiger um ein Element weiter** in Richtung Dateiende.

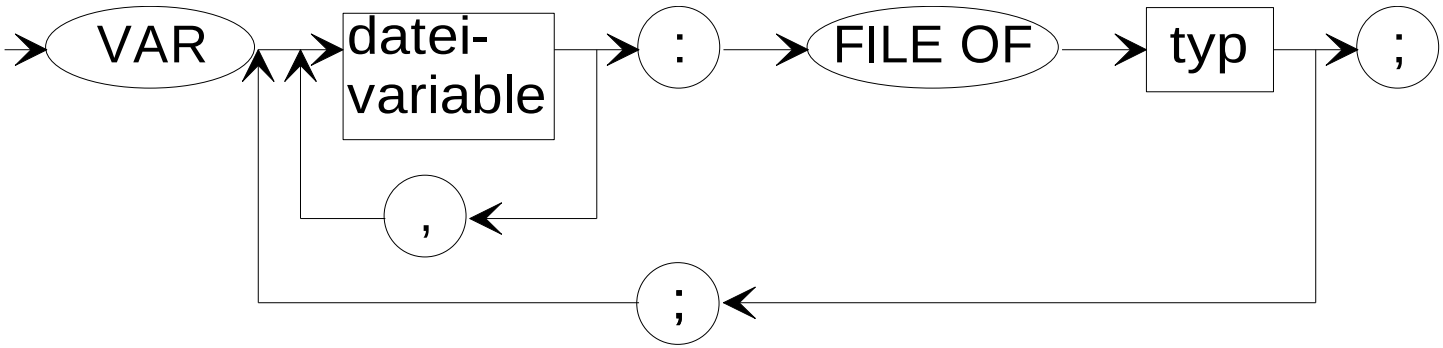
Deklaration der Dateivariablen

Mit der **Dateivariablen** wird ein logischer Dateiname festgelegt. Unter dieser Dateivariablen werden alle Operationen auf die Datei ausgeführt. Bei der Festlegung der Dateivariablen wird gleichzeitig der Typ spezifiziert.

Deklaration der Dateivariablen:

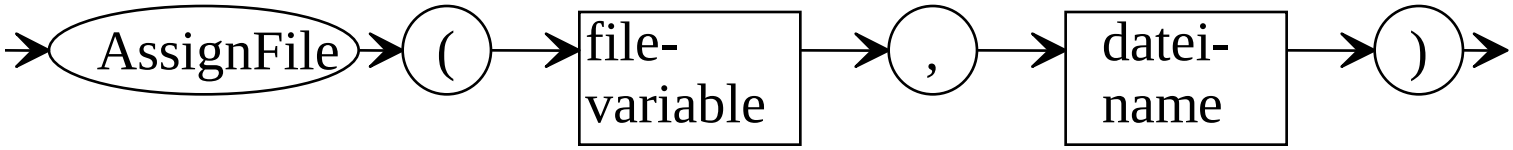
Beispiel:

```
VAR dat1,dat2 : file of integer;
    fi1       : file of real;
```



Verknüpfen von logischer und physischer Datei

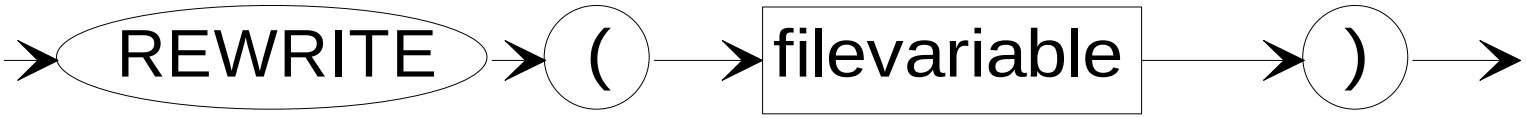
Um den Zugriff auf eine physische Datei zu ermöglichen, muss diese mit der logischen Datei, deren Name in der Dateivariablen festgelegt wurde, verknüpft werden. Dazu dient die Dateizuweisung **AssignFile**. Der Name der physischen Datei ist dabei ein String, der die Form einer Variablen oder Konstanten haben kann. Der physische Dateiname wird der logischen Dateivariablen zugewiesen. Alle auf die Filevariable ausgeübten Funktionen beziehen sich auf das physische (Disketten- oder Festplatten-) File.



Aus Kompatibilitätsgründen ist der Zuordnungsbefehl Assign erhalten geblieben.

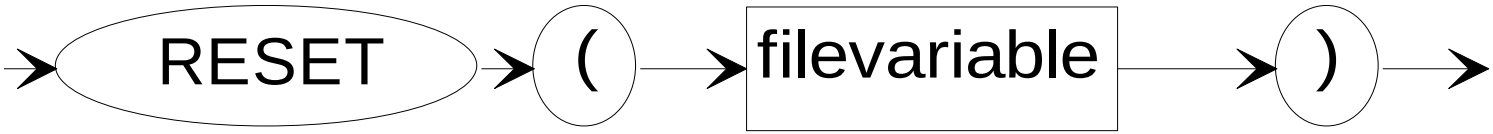
Öffnen einer neuen Datei

Die Prozedur **Rewrite** dient zur **Eröffnung einer neuen Datei**, deren Name in der ASSIGN-Anweisung festgelegt wird. Der Dateizeiger der neuen Datei wird auf das 0. Element gesetzt und diese Datei damit eröffnet. Existiert schon eine Datei mit diesem Namen, so wird diese gelöscht und anschließend als leere Datei neu erstellt. Der Dateizeiger steht nach dem REWRITE auf dem 0. Element, welches EOF ist. Bei Textdateien wird die Datei für Schreiboperationen, bei typisierten Dateien für Schreib- und Leseoperationen geöffnet. Wird Rewrite auf eine geöffnete Datei angewendet, so wird diese geschlossen, gelöscht und neu angelegt.



Öffnen einer bestehenden Datei

Die Prozedur **Reset** setzt den Dateizeiger einer bereits bestehenden Datei auf das 0. Element und eröffnet damit diese Datei. Ist die Datei bereits eröffnet, so wird der Datenzeiger auf das 0. Element der Datei gesetzt.
Existiert die Datei noch nicht, so wird die Abarbeitung mit einem Laufzeitfehler beendet.

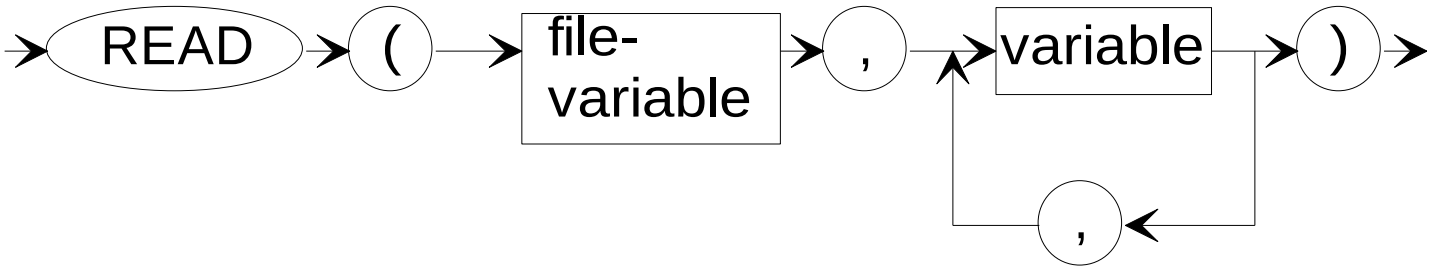


Öffnen einer bestehenden Datei unter Nutzung der Opendialogkomponente

```
procedure p_dateioeffnen;
 var v_ausgewaehlt : boolean;
 begin
  V_ausgewaehlt:=form1.opendialog_datei.execute;
  if v_ausgewaehlt then
      begin
      V_datname := form1.opendialog_datei.FileName;
      assignfile(v_datei,v_datname);
{$i-} reset(v_datei);
      if ioresult <> 0 then
         rewrite(v_datei);
{$i+} v_datinit := true;
      end;
 end; // of procedure p_dateioeffnen
```

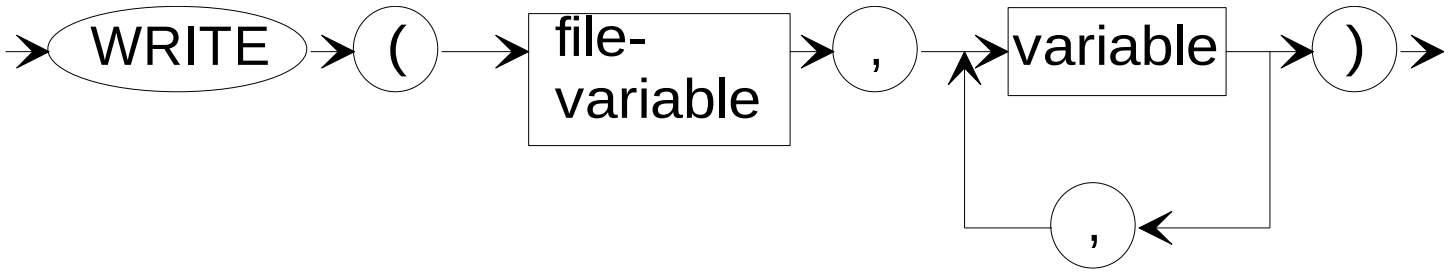
Lesen aus einer Datei

Die Prozedur **READ** liest eine oder mehrere Komponenten einer typisierten Datei an der Position des Dateizeigers in die angegebenen Variablen. Der Dateizeiger wird um die Anzahl der gelesenen Komponenten weitergerückt



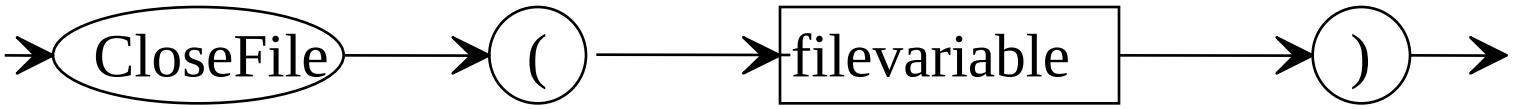
Schreiben in eine Datei

Zum Schreiben von Daten in die Datei wird die Prozedur **WRITE** verwendet. Durch die Prozedur **WRITE** werden nacheinander die Inhalte der Variablen in die Komponenten der durch die Dateivariable zugeordneten Datei geschrieben. Begonnen wird mit der Komponente, auf die der Dateizeiger weist. Nach jeder Zuweisung wird der Dateizeiger auf die nächste Komponente eingestellt. Weist der Dateizeiger vor Beginn einer Schreibaktion auf das Dateiende , so wird die Datei durch die Schreibaktionen erweitert. Nach der Schreibaktion befindet sich der Dateizeiger wieder auf dem Dateiende.



Schließen einer Datei

Nach Beendigung der Dateioperationen muss die Datei geschlossen werden. Dazu dient die Prozedur **CloseFile**. Es wird die Datei geschlossen, die der Dateivariablen mit AssignFile zugeordnet wurde. Das physische File wird geschlossen, und der aktuelle Dateistatus wird in das Datenträgerverzeichnis eingetragen. Wird versucht, eine Datei zu schließen, die vorher nicht eröffnet wurde, so wird die Abarbeitung mit einem Laufzeitfehler abgebrochen (erst ab TURBO-Pascal-4). Die Verbindung zwischen Dateivariable und Dateiname wird nicht aufgehoben.



Aus Kompatibilitätsgründen ist die (überladene) Prozedur Close weiterhin nutzbar.

Bestimmen der Dateigröße

Die Größe einer Datei kann mit der Funktion **FILESIZE** ermittelt werden. Diese Funktion liefert die Anzahl der Komponenten, die in der Datei gespeichert sind.
Diese Funktion ist nicht für Dateien des Typs Textfile anwendbar. Sie kann nur auf geöffnete Dateien angewendet werden.

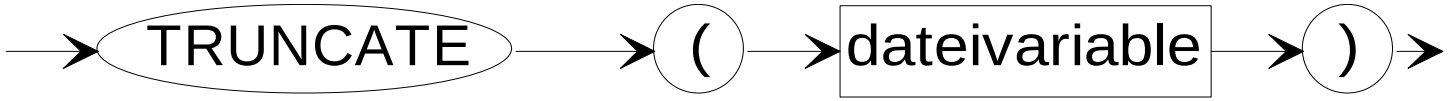
Beispiel:

```
a := FILESIZE( dat );
```

Abschneiden einer Datei

Eine weitere Prozedur zur Arbeit mit dem Dateizeiger ist die Prozedur **TRUNCATE**. Diese Prozedur schneidet eine Datei an der aktuellen Position des Dateizeigers ab.

Diese Prozedur ist nicht für Dateien vom Typ Textfile anwendbar. Alle Daten hinter dem Dateizeiger werden beim Aufruf dieser Prozedur gelöscht. Die Position des Dateizeigers ist anschließend EOF.



Prüfung ob eine Datei vorhanden ist

Mit der Funktion **FileExists** kann überprüft werden, ob eine Datei vorhanden ist.
FileExists gibt True zurück, wenn die im Parameter FileName angegebene Datei vorhanden ist. Existiert die Datei nicht, wird False zurückgegeben.
Die Besonderheit dieser Funktion besteht darin, das sie **direkt auf den Dateinamen angewendet** wird.

```
procedure p_datauf_lesen;
   var datname : string;
       existens : boolean;
  begin
    repeat
    datname :=inputbox('Dateiarbeit','Bitte
Dateinamen eingeben','');
    existens := Fileexists(datname);
.........
```

Löschen einer Datei

Mit der Prozedur **ERASE** besteht die Möglichkeit, eine bestimmte Datei zu löschen. Es wird die Datei gelöscht, die der Filevariablen mit AssignFile zugewiesen wurde. Die Datei darf nicht offen sein. Wurde auf die Datei ein RESET oder REWRITE ausgeführt, so muss die Datei vor der Anwendung von ERASE mit CloseFile geschlossen werden.

ERASE löscht die Datei ohne Rückfrage.

Die Unit System muss eingebunden sein.

Umbenennen einer Datei

Mit der Prozedur **RENAME** besteht die Möglichkeit, eine Datei neu zu benennen und den Dateieintrag bei Bedarf in ein neues Verzeichnis zu setzen. Es wird die Datei umbenannt, die der Filevariablen in der AssignFile-Anweisung zugewiesen wurde. Der neue Name wird in der Zeichenkette newname angegeben. Dieser Name kann einen Pfadnamen enthalten. Die umzubenennende Datei darf nicht geöffnet sein.

Die Unit System muss eingebunden sein.

Grundoperationen auf Dateien – Datensätze anhängen

Zum Anhängen eines Elementes wird der Dateizeiger hinter das letzte Element positioniert (EOF). Der nachfolgende Schreibvorgang führt alle notwendigen Zuordnungs- und Verwaltungsoperationen aus.

```
reset(v_datei);
  while not(eof(v_datei)) do
    read(v_datei,hsatz);
  write(v_datei,satz);
```

Positionieren auf 0. Element

Lesen bis zum Dateiende

Schreiben des neuen Satze hinter das Dateiende

Grundoperationen auf Dateien – Löschen von Datensätzen

Das Löschen von Datensätzen kann nur durch ein Verkürzen der Datei realisiert werden. Dazu wird die Prozedur **TRUNCATE** verwendet.

Sollen Datensätze innerhalb der Datei gelöscht werden, so muss über Schreib-Leseoperationen eine Verschiebung der nachfolgenden Datensätze erfolgen. Nach Abschluss dieser Operationen wird die Datei um einen Datensatz gekürzt.

Grundoperationen auf Dateien – Navigieren in Dateien

Zur Navigation in Dateien wird die Prozedur **seek** verwendet. Um sicherzustellen, dass der Dateizeiger in den implementierten Prozeduren auf das angezeigte Element positioniert wurde, ist es erforderlich, nach dem Lesen des Datensatzes eine Rückpositionierung um einen Datensatz vorzunehmen.

```
position := filepos(v_datei);
if position > 0 then
   position := position - 1;
seek(v_datei,position);
read(v_datei,v_adresse);
p_anzeigen(v_adresse);
seek(v_datei,filepos(v_datei)-1);
```

aktuelle Position

Rückpositionieren, Lesen, Anzeigen

Rückpositionieren nach Lesevorgang

Grundoperationen auf Dateien – Löschen von Datensätzen

```
seek(v_datei,filepos(v_datei)-1);
   auswahl := MessageDlg('Datensatz wirklich
löschen?',mtConfirmation,
   [mbYes, mbNo], 0);
   aktposition := filepos(v_datei);
   laenge := filesize(v_datei);
   if auswahl = mryes then
   begin
     for i := aktposition to laenge-2 do
       begin
         seek(v_datei,i+1);
         read(v_datei,hsatz);
         seek(v_datei,i);
         write(v_datei,hsatz);
       end;
       seek(v_datei,laenge-1);
       truncate(v_datei);
       seek(v_datei,aktposition);
```

Positionieren des zu löschenden Elementes

Lücke an das Dateiende schieben

Lücke Löschen

Grundoperationen auf Dateien – Einfügen von Datensätzen

Sollen Datensätze an beliebigen Positionen eingefügt werden, so ist dies nur indirekt möglich. Werden Datensätze eingefügt, so muss der notwendige Platz durch Verschieben geschaffen werden.

Einfügen eines Datensatzes vor der aktuellen Position

```
while not(eof(v_datei)) do
  begin
  read(v_datei,hsatz);
  seek(v_datei,filepos(v_datei)-1);
  write(v_datei,satz);
  satz := hsatz;
  end;
 write(v_datei,satz);
```

einzufügender Satz

Verschieben der Nachfolgesätze

Lesen in einer Textdatei mit Read

Die Prozedur **READ** liest eine Anzahl von Werten aus einer Textdatei in die angegebenen Variablen.

CHAR-Typ

Ein Zeichen wird aus der Datei gelesen, der Positionszeiger wird um ein Zeichen weitergesetzt.

INTEGER-Typ

READ erwartet eine Folge von numerischen Zeichen. Führende Leerzeichen, TABs und Zeilenvorschübe werden übergangen.

Das erste nichtnumerische Zeichen oder Dateiende bricht den Lesevorgang ab. Ergibt die eingelesene Zeichenfolge einen Wert im erwarteten Format, so wird dieser Wert der Variablen zugewiesen. Falls sich kein Wert im erwarteten Format ergibt, wird ein Laufzeitfehler erzeugt. Wird vor dem Lesen eines numerischen Wertes das Dateiende erreicht, so erhält die Variable den Wert 0.

Lesen in einer Textdatei mit Read

REAL-Typ

Wie bei Integer-Variablen erwartet READ eine Folge von numerischen Zeichen. Zusätzlich sind auch Dezimalpunkt und E möglich. Das Leseverhalten ist wie bei Integer-Variablen.

STRING-Typ

READ liest Zeichen bis zum Erreichen des Zeilenendes. Diese Zeichen werden der STRING-Variablen zugeordnet. Ist die Zeile länger als 255 Zeichen, so werden nur die ersten 255 Zeichen in die Variable übertragen. Der Rest der Zeichen wird ignoriert.

Der Dateizeiger wird auf das Zeilenende positioniert, ohne das Zeilenendezeichen zu lesen. Das nächste READ würde dieses Zeilenendezeichen lesen. Daraus folgt, dass aufeinanderfolgende Leseaktionen mit String-Variablen beim zweiten Aufruf immer einen Nullstring liefern.

Lesen in einer Textdatei mit Readln

Die Prozedur **READLN** liest eine Anzahl von Werten aus einer Textdatei in die angegebenen Variablen und geht dann zum Anfang der nächsten Zeile in der angegebenen Datei. Wird keine Variable angegeben, so geht READLN zur nächsten Zeile über und ignoriert dabei die gelesenen Zeichen.
Der Unterschied zu READ besteht darin, dass READLN immer bis zum Zeilenende liest. Stehen mehr Elemente in der Zeile, als Variablen angegeben sind, so werden die restlichen ignoriert. Werden mehr Variablen angegeben, als Elemente in der Zeile stehen, so bleiben die restlichen Variablen leer bzw. erhalten den Wert 0.

Die Behandlung der einzelnen Datentypen erfolgt analog READ.

Schreiben in einer Textdatei

WRITE

Die Prozedur WRITE schreibt die Daten in die angegebene Datei. Es können neben Variablen auch Ausdrücke ausgegeben werden. Dabei besteht die Möglichkeit Ausgabeformate zu nutzen.

WRITELN

Die Prozedur WRITELN führt WRITE aus und schließt die Ausgabe mit Zeilenvorschub (CR-LF) ab. Writeln darf nur auf Textdateien angewendet werden. Writeln auf die Ausgabedatei führt zu einem Zeilenvorschub.

Schreiben einer Textdatei mit *savetofile*

Einige Dialogkomponenten bieten die Möglichkeit, direkt in eine Textdatei zu schreiben, bzw. Texte direkt in die Dialogkomponente zu laden.

```
procedure TForm1.Button1Click(Sender: TObject);
begin

form1.listbox1.items.savetofile('protokoll.txt');
end;
```

Kontrollfragen

1. Erläutern Sie die grundlegende Struktur einer Datei zur Verarbeitung von Daten in Object-Pascal. Worin unterscheiden sich Textdateien von typisierten Dateien?
2. Beschreiben Sie die notwendigen Schritte zum Schreiben und Lesen in typisierten Dateien in Object-Pascal. Gehen Sie dabei auf die unterschiedlichen Möglichkeiten zum Öffnen von Dateien ein. Welche Funktion hat die Dateivariable? Wie kann der Dateizeiger auf einen vorgegebenen Datensatz positioniert werden?
3. Beschreiben Sie die notwendigen Schritte zur Verarbeitung von Textdateien in Object-Pascal. Gehen Sie dabei auf die unterschiedlichen Möglichkeiten zum Öffnen von Dateien ein. Worin bestehen beim Schreiben und Lesen von Daten die Unterschiede zu typisierten Dateien?
4. Welche Methoden bietet Borland-Delphi, Daten direkt aus Komponenten, z.B. der Memokomponente, zu laden und zu speichern. In welchem Format werden die Daten abgelegt?