

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

Duale Hochschule Baden Württemberg
Stuttgart
University of
Cooperative Education

Programmieren in Pascal und Delphi

Der Deklarationsteil

Pascal trennt, anders als andere Programmiersprachen, strikt Deklarationen und Befehle. Eine Deklaration betrifft die Daten und Befehle repräsentieren den Code. Diese Trennung spiegelt auch die Architektur des Prozessors wieder, der unterscheidet zwischen einem Datensegment und einem Codesegment.

Der Deklarationsteil kann an drei Stellen vorkommen:

- In einer Unit im Interface Teil: Alle hier deklarierten Namen und Typen stehen allen Programmen zur Verfügung, welche diese Unit nutzen. Wir werden dazu noch zurückkommen, wenn wir die Struktur von Units besprechen.
- Im Implementationsteil einer Unit: Diese Daten sind nur lokal innerhalb dieser Unit bekannt, aber hier für alle Funktionen.
- Innerhalb einer Funktion: Hier sind sie nur in dieser Funktion und ihren Unterfunktionen bekannt.

Beziehen sich Daten aufeinander, z.B. auf Vereinbarungen, so gilt: Der Quelltext wird sequentiell gelesen. Es muss also ein Wert definiert sein, bevor sein Name verwendet werden kann. Daher gibt es eine Reihenfolge für den Deklarationsteil: Zuerst Konstanten, dann Typen und dann Variablen. Man kann jedoch dies mischen, also mehrere Definitionen von Konstanten und Typen machen.

Konstanten

Konstanten sind Werte, die nicht verändert werden können. Sie werden oft benutzt um Dimensionen von Arrays, unabänderliche Daten, wie der Titel des Programms oder Ähnliches festzulegen.

In der Regel sind die Konstanten der erste Teil der Deklaration, da sie auf nichts basieren, aber benötigt werden, für andere Definitionen.

Die Definition von Konstanten geschieht durch das Schlüsselwort „**Const**“ gefolgt von einer Liste von Konstanten in der Form:

Konstantenname = Wert;

oder

Konstantenname = Ausdruck;

Ein Datentyp ist nicht nötig, Pascal ermittelt den korrekten Datentyp automatisch. Ein Ausdruck kann ein mathematischer Ausdruck ($4 \cdot \text{Pi}$, $100 \cdot 4$) oder jeder andere Ausdruck sein, der zur Übersetzungszeit ausgewertet werden kann: Alle Informationen zur Berechnung müssen dann vorliegen.

Beispiel:

```
// einfache Konstanten
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
Const
Autor = 'Bernd Leitenberger';
Version = 1.23;
Datum = 2009;
Max = 1000;
// Konstanten, gebildet aus Ausdrücken
Pi2 = 2*Pi;
Version2 = Version+1;
Autor_homepage = Autor+' (www.bernd-leitenberger.de)';
min = max-900;
```

Es ist möglich eine Konstante auf einer anderen basierend zu deklarieren (hier z.B. bei min erfolgt), wenn diese weiter oben deklariert ist.

Bei der Übergabe von Parametern kann auch die Deklaration einer Konstante erfolgen. Das bedeutet, dass innerhalb der Prozedur der Wert nicht verändert werden kann. Übergeben kann aber nicht nur eine Konstante, sondern auch eine Variable werden. Für große Variablen hat dies den Vorteil, dass sie nicht kopiert werden müssen, wie bei den Wertparametern. Dazu wird dem Parameter das Wort **const** vorangestellt.

Es gibt noch eine spezielle Form der Variablen, es sind die beschreibbaren Konstanten, besser gesagt, es sind Variablen, die vorgelegt sind. Es gibt zwei Möglichkeiten der Deklaration: einmal in der Konstantendeklaration in der Form:

Const

Konstantenname : Datentyp = Wert;

Die zweite Möglichkeit der Definition ist in der Variablensektion in derselben Form:

var

Konstantenname : Datentyp = Wert;

In der Benutzung gibt es aber einen Unterschied. Die Definition als Variable ist nicht erlaubt in Funktionen und Prozeduren. Die Definition als Konstante ist älter und hat nicht diesen Nachteil, aber in der Voreinstellung liefert sie einen Compilerfehler, denn man jedoch abschalten kann.

Die beschreibbaren Konstanten haben einen großen praktischen Nutzen, denn in der Praxis müssen die meisten Variablen mit einem Startwert versehen werden. Dies kann man sich so sparen.

Typen

Bislang haben wir schon einige Datentypen kennengelernt. Dies wird nun durch weitere ergänzt. Pascal bietet aber auch die Möglichkeit eigene Datentypen zu definieren, die auf den internen Datentypen basieren. Die Typendefinition folgt meistens der Konstantendefinition. Sie beginnt mit dem Schlüsselwort **Type**, gefolgt von einer Liste in der Form:

Datentypname = Beschreibung des Datentyps;

Hier einige Beispiele:

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

Type

```
Unterbereichstyp = 1..1000;  
Aufzaehlungstyp = (Student, Professor, Dozent);  
DefAufzaehlungstyp =  
(Einstiegsgehalt=100, Praemiengehalt=200, Beamtenpension=1000);  
Mengentyp = Set of Aufzaehlungstyp;
```

Diese Auflistung zeigt die drei Typen, die wir heute kennenlernen werden.

Der Teilbereichstyp

Der Unterbereichstyp grenzt den Bereich eines ordinalen Typs ein. (Zahlen, Zeichen oder ein Aufzählungstyp s.u.) Im obigen Beispiel wurde vom Typ Integer nur der Bereich 1 bis 1000 benutzt. Die Definition ist folgende:

Type

<Typname> = Wert1 .. Wert2;

Der Bereich wird zwischen Wert1 und Wert2 mit zwei Punkten (..) angegeben. Der Nutzen liegt auf der Hand. Man kann diesen Typ zur Angabe einer Array Dimensionierung nutzen und auch für alle Variablen, die als Indizes benutzt werden. Dies zeigt folgende Demonstration:

Type

```
TIndex = 1..1000;  
TArray = Array [TIndex] of Double;  
  
function ArrayDemo : TArray;  
  
var  
  aArray : TArray;  
  i      : TIndex;  
  
begin  
  for i:=Low(TIndex) to High(TIndex) do  
    aArray[i]:=Random;  
  Result:=aArray;  
end;
```

Der Vorteil dieses Typs ist, dass man den Compiler instruieren kann, während des Programmlaufs eine Verletzung des Bereichs zu erkennen (Compilerschalter `{SR+}`). Bei Ausdrücken, die zur Übersetzungszeit feststehen wie z.B.

```
i := -5;
```

wird dies schon bei der Übersetzung als Fehler markiert. Für die Definition der Grenzen sollte man Konstanten benutzen, oder man benutzt - wie hier - die Funktionen `High()` und `Low()`, welchen das kleinste oder größte Element eines ordinalen Typs liefern. In diesem Fall den Startindex und den Index des letzten Elements.

Der Aufzählungstyp

Dieser Typ besteht aus einer Aufzählung von verschiedenen Werten. Eine Variable kann jeweils einen dieser Werte einnehmen. Man benutzt ihn vor allem, um Programme lesbarer zu machen, da die einzelnen Werte praktisch Konstanten entsprechen, aber eben nur diese Konstanten verwendet werden können.

Ein weiterer Einsatz liegt darin, Zustände zu speichern. Also eine Auswahl aus mehreren Möglichkeiten. Denken sie an einen Text: Der Text kann linksbündig, rechtsbündig, zentriert oder im Blocksatz formatiert sein – mehr Möglichkeiten gibt es nicht. Oder es könnte die letzte Aktion des Benutzers (für einen Rückgängig-Befehl) gespeichert werden: Auch hier ist die Auswahl begrenzt auf die verfügbaren Menübefehle.

Die Definition geschieht im Typendeklarationsteil nach dem Stichwort `type` in der Form:

type

<Typenname> = (Wert1,Wert2,Wert3);

Hier einige Beispiele:

Type

```
Aufzaehlungstyp      = (Student, Professor, Dozent);
DefAufzaehlungstyp =
(Einstiegsgehalt=100, Praemiengehalt=200, Beamtenpension=1000);
```

procedure Demo_Typen;

var

```
ba_angehoerige: Aufzaehlungstyp;
gehalt         : DefAufzaehlungstyp;
zahl           : integer;
```

begin

```
ba_angehoerige:= Student;
zahl:=Ord(Professor); // zahl = 1
ba_angehoerige:=Aufzaehlungstyp(2);
gehalt:=Praemiengehalt;
zahl:=Ord(gehalt); // zahl = 200
```

end;

Intern werden die Aufzählungstypen als Integer Variablen gespeichert, wobei der erste Wert als 0 gespeichert wird, der Zweite als 1 usw. Durch Zuweisen eines Wertes bei der Definition kann man dies ändern. Vorsicht: Ein Wert kann hier auch doppelt zugewiesen werden. Das gibt dann keinen Fehler, nur haben zwei Konstanten denselben Wert.

Die Konvertierung eines Aufzählungstyps in einen anderen ordinalen Wert (eine Integerzahl) kann mit der Funktion `ord()` erfolgen. Ein ordinaler Wert kann über einen sogenannten **Typecast** erfolgen: Jeder ordinale Wert liefert seinen internen Wert als Integer zurück, wenn man die **Typdefinition als Funktion** benutzt: In diesem Falle `Aufzaehlungstyp(2);` oder `zahl:=Ord(gehalt);`

Der Mengentyp

Der Mengentyp ähnelt dem Aufzählungstyp. Der Unterschied liegt darin, dass ein Mengentyp **mehrere** mögliche Werte gleichzeitig einnehmen kann. Die Definition erfolgt in der Form:

type

<Typname> = SET OF <exisiterender einfacher Typ>

Es gibt für den Mengentyp folgende drei elementare Operationen:

- + Bildet die Vereinigungsmenge von zwei Mengen
- Bildet die Schnittmenge von zwei Mengen

IN stellt fest, ob ein Element Bestandteil der Menge ist.

Die Definition einer Menge geht in der Form **Set of** Ordinaltyp; Ordinaltyp heißt: Die Elemente müssen abzählbar sein, also ganze Zahlen, Char aber auch die oben schon erwähnten Teilbereich und Aufzählungstypen. Es ist möglich einen Mengentyp auf einem vorher definierten Aufzählungstypen zu definieren.

Die einzelnen Werte werden in eckige Klammern [] gesetzt. Die beiden Punkte geben einen Bereich an. Einzelne Werte können auch durch Kommas getrennt werden. So hat die Variable „zahlen“ die Werte „0“, aber auch 1,2,3,4,5,6,7,8,9. Die leere Menge gibt man durch zwei Klammern an [].

Type

```
Zeichentyp      = Set of Char;
```

var

```
zahlen          : Zeichentyp;  
Buchstaben      : Zeichentyp;  
Grossbuchstaben: Zeichentyp;  
Kleinbuchstaben: Zeichentyp;  
Pascalzeichen  : Zeichentyp;  
Zahlen_ab_4     : Zeichentyp;  
Zeichen         : Char;
```

begin

```
zahlen:=['0'..'9'];  
Buchstaben:=['a'..'z','A'..'Z'];  
Grossbuchstaben:=['A'..'Z'];  
Kleinbuchstaben:=Buchstaben-Grossbuchstaben;  
Pascalzeichen:=['_']+Buchstaben+Zahlen;  
zahlen_ab_4:=Zahlen - ['0'..'3'];  
write('Geben sie ein Zeichen ein: '); Readln(zeichen);  
write('Das Zeichen ',zeichen,' ist Bestandteil der folgenden Mengen: ');  
if zeichen in Zahlen then write(' Zahlen ');  
if zeichen in Buchstaben then write(' Buchstaben ');  
if zeichen in Grossbuchstaben then write(' Grossbuchstaben ');  
if zeichen in Kleinbuchstaben then write(' Kleinbuchstaben ');  
if zeichen in Pascalzeichen then write(' Pascalzeichen ');  
if zeichen in Zahlen_ab_4 then write(' Zahlen ab 4 ');  
writeln;  
end.
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

Dies zeigt auch die Verwendung der Operationen. Sehr oft wird der Mengentyp ohne Typendeklaration, einfach durch Angabe einer Menge benutzt um Wertebereiche zu prüfen:

```
var wahl : Char;  
  
begin  
if wahl in ['A','C','M','9'] then Process_Auswahl(wahl);
```

In diesem Fall wird die Prozedur `Process_Auswahl` nur abgesprungen, wenn `Wahl` einen der angegebenen Werte hat.

Es gibt eine besondere Form der For Schleife mit dem Schlüsselwort `IN`:

FOR <Variable> IN <Mengentyp> DO

wird Bei jedem Durchlauf der Schleife der Variable das nächste Element in der Menge zuweisen (Mengen sind nicht wie Arrays indizierbar also man kann nicht auf ein Element über einen Index [] zugreifen).

Mengen werden selten gebraucht, jedoch ist das Prüfen mit `IN` ob eine Zahl in einem bestimmten Bereich ist sehr häufig und erspart eine verschachtelte Verknüpfung von `OR`'s wie z.B. das Prüfen auf Buchstaben + Umlaute zeigt:

```
if buchstabe in ['A'..'Z','a'..'z','ä','ö','ü','ö','Ä','Ü','ß'] then
```

ist gleichbedeutend mit:

```
if (buchstabe>='a' and buchstabe<='z') or (buchstabe>='A' and buchstabe<='Z') or  
(buchstabe='ö') or (buchstabe='Ö') or (buchstabe='ä') or (buchstabe='Ä') or  
(buchstabe='ü') or (buchstabe='Ü') or (buchstabe='ß') then
```

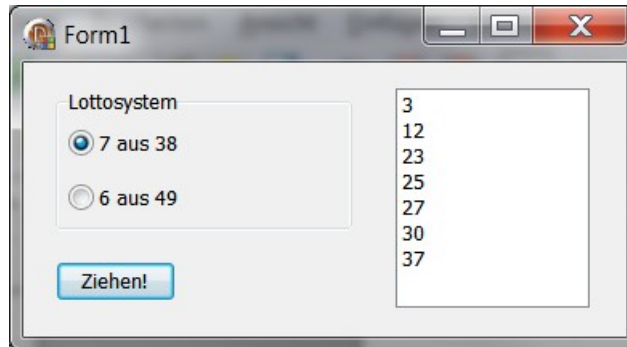
Manche Problemstellungen sind mit Mengentypen sehr elegant zu lösen: Hier ein grafisches Programm zum Ziehen der Lottozahlen:

```
procedure TForm1.Button1Click(Sender: TObject);  
  
type  
lottoindex = 1 .. 49;  
  
var  
lottozahlen: set of lottoindex;  
anzahl, max, zahl: integer;  
  
begin  
if RadioGroup1.ItemIndex = 0 then max := 7 else max := 6;  
anzahl := 0;  
lottozahlen := []; // Leere Menge  
repeat  
repeat  
if max = 7 then  
zahl := random(38) + 1  
else  
zahl := random(49) + 1;  
until not(zahl in lottozahlen); // solange Zahl ziehen bis sie noch nicht gezogen  
wurde  
lottozahlen := lottozahlen + [zahl]; // Set erhöhen  
inc(anzahl);
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
until max = anzahl;  
ListBox1.Items.Clear;  
for zahl in lottozahlen do ListBox1.Items.Add(inttostr(zahl));  
end;
```

Dazu gehört folgendes Formular:



Ohne eine Sortierfunktion gibt die FOR IN Schleife schon die Zahlen sortiert aus. Wenn sie den Aufwand mit der Lösung mit Arrays aus dem ersten Semester. Hier ein Beispiel für die Verwendung von Strings, Mengentypen und Teilbereichstypen:

```
program Buchstaben_zahlen;
```

```
 {$APPTYPE CONSOLE}
```

```
uses
```

```
  SysUtils;
```

```
type
```

```
  Grossbuchstabenindex = 'A'..'Z'; // Definition eines Teilbereichstyps
```

```
  Grossbuchstabenarraytyp = Array [Grossbuchstabenindex] of integer;
```

```
  // Deklaration eines Arraytyps auf dem obigen Teilbereichstyp
```

```
Var Quellstring : String; // Eingabetext
```

```
  Grossbuchstabenarray : Grossbuchstabenarraytyp; // Array, das das Vorkommen jedes
```

```
  Buchstabens zählt
```

```
  Grossbuchstaben : Set of Char; // benötigt zum Prüfen, ob ein Zeichen auch ein
```

```
  Buchstabe ist
```

```
  zeichen,c : char; // Variablen um ein Zeichen des Strings aufzunehmen und durch das
```

```
  array zu iterieren
```

```
  i : Cardinal; // Benötigt um durch die Länge des Quellstrings zu iterieren
```

```
begin
```

```
  Grossbuchstaben:=['A'..'Z']; // Definition der Menge der Buchstaben
```

```
  write('Geben Sie einen Text ein: ');
```

```
  Readln(Quellstring);
```

```
  for c:=low(Grossbuchstabenarray) to high(Grossbuchstabenarray) do
```

```
Grossbuchstabenarray[c]:=0;
```

```
  // Initialisierung des Zählarrays
```

```
  for i:=1 to length(Quellstring) do // Iterieren über den Eingabestring
```

```
  begin
```

```
    zeichen:=Uppcase(Quellstring[i]); // Kleinbuchstaben in Großbuchstaben umwandeln
```

```
    if zeichen in Grossbuchstaben then
```

```
      Grossbuchstabenarray[Zeichen]:=Grossbuchstabenarray[Zeichen]+1;
```

```
      // Ist das Zeichen Bestandteil der Großbuchstabenmenge, dann erhöhe den Zähler für
```

```
      dieses Zeichen um 1
```

```
    end;
```

```
  // Ausgeben, iterieren über das Zählarray
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
for c:=low(Grossbuchstabenarray) to high(Grossbuchstabenarray) do
if Grossbuchstabenarray[c]>0 then // Nur Ausgabe, wenn auch Buchstabe vorkommt
writeln('Buchstabe ',c,' kommt ',Grossbuchstabenarray[c],' mal vor');
readln;
end.
```

Aufgabe

Erzeugen sie ein leeres Formular. Schauen sie im Objektinspektor sich die Eigenschaft an. Welche der Eigenschaften haben den Dateityp:

- Mengentyp
- Aufzählungstyp
- Indextyp

Die Lösung:

Aufzählungstyp: Align, Bordericons,

Mengentyp: Anchors, BiDiMode, Borderstyle, Cursor, Defaultmonitor, Dragkind, Dragmode, Formstyle, Helpstyle, Popupmode, Position, Printstyle, Windowstate

Teilbereichstyp: Nicht vertreten.

Records

Records sind der letzte Datentyp, den wir noch kennen lernen. Das besondere von Records (Datensätzen) ist, dass sie kein neuer Datentyp an sich sind, sondern sie Daten verschiedener Variablen bündeln. Nehmen sie die Daten, die zu einem Konto gehören: Dies könnten Kontonummer, Name des Kontoinhabers und Kontostand sein. Diese Daten gehören zusammen. Anstatt drei Variablen zu definieren, definiert man einen neuen Datentyp und eine Variable dieses Typs:

```
type
  Kontorecord = record
    Kontonummer : integer;
    Kontostand  : double;
    Kontoinhaber: string;
  end;
var konto : Kontorecord;
```

Darin sieht man auch die allgemeine Definition eines Rekords. Als Datentyp erfolgt diese in der **Type** Sektion. Sie beginnt mit:

<datenfeldname> = record

Und endet mit einem:

end;

Dazwischen befinden sich alle Datenfelder, also Subvariablen, wie sie in der Variablendeklaration erfolgen würde, also mit:

<variablenname> : <datentyp>

Wie bei jedem Typ ist dies nur die Definition. Zum Benutzen muss dazu noch eine Variable dieses Typs (der Rekordname) in der VAR-Sektion definiert werden. In diesem Falle ist die Variable Konto vom Datentyp Kontorecord.

Auf die einzelnen Datenfelder eines Rekords kann man mit der Form:

<Recordname>.<datenfeld>

Zugreifen, also mit einem Punkt dazwischen. Der Rekordname ist der Name der Variable (in der VAR-Sektion) definiert und das Datenfeld ist der Name eines Feldes der Rekorddefinition (in der Type Sektion definiert).

```
konto.kontonummer:=26757890;
konto.kontostand:=konto.kontostand+47.11; // 47.11 Euro eingezahlt
writeln('Inhaber: ',konto.kontoinhaber);
```

Es geht aber auch bequemer. Mit der Anweisung „**with**“ kann man einen Satz öffnen und muss dann nur noch die Datenfelder angeben. Der Rekordname kann weggelassen werden:

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
with <reordname> do
begin
  <code der auf die Felder zugreift>
end;
```

Recordname ist der einer existierenden Variable des Datensatzes (nicht des Typs!). Die obigen drei Zeilen kann man kurz auch so schreiben:

```
with Konto do
begin
  Kontonummer:=26757890;
  kontostand:=kontostand+47.11; // 47.11 Euro eingezahlt
  writeln('Inhaber: ',konto.inhaber);
end;
```

Der wichtigste Vorteil eines Rekords ist aber, dass er selbst wiederum eine einzige Variable mit dem Inhalt aller Subvariablen ist. Er kann so einfach kopiert werden:

```
type
  Kontorecord = record
    Kontonummer : integer;
    kontostand : double;
    konto.inhaber: string;
  end;

var konto, kontoneu : Kontorecord;

begin
  konto.Kontonummer:=26757890;
  konto.kontostand:=konto.kontostand+47.11; // 47.11 Euro eingezahlt
  writeln('Inhaber: ',konto.konto.inhaber);
  kontoneu:=konto;
  kontoneu.Kontonummer:=3733774;
end.
```

„Kontoneu“ enthält nach der Zuweisung von Konto alle Daten, mit der nächsten Anweisung wird dann die Kontonummer von Kontoneu geändert (Die bei „Konto“ bleibt natürlich so, wie sie vorher war). Bei drei Variablen für Kontonummer, Kontostand und Konteninhaber wären dagegen drei Zuweisungen nötig.

Rekords sollten sie immer verwenden, wenn Daten zusammengehören und einzeln keinen Sinn machen: Adressen, Bestellungen, Produktdaten. So können sie insbesondere Records in Arrays speichern:

```
type
  Kontorecord = record
    Kontonummer : integer;
    kontostand : double;
    Konto.inhaber: string;
  end;

var
  konto: Kontorecord;
  karray: array [1..100] of Kontorecord;

begin
  [...] // anderer Code
  karray[i]:=konto; // record ins Array kopieren
  konto.kontonummer:=433665; // Record ändern
  karray[i+1]:=konto; // geänderten Record erneut speichern
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
konto:=karray[i-1]; // Record aus dem Array auslesen.  
karray[i].kontoinhaber:='Uwe Müller';  
// Zugriff auf ein Feld eines Records in einem Array  
[...]
```

Typisierte Dateien

Alle Daten haben nur einen Sinn, wenn man sie auch speichern kann. Es gibt bei Delphi sehr verschiedene Typen von Dateien. Sie haben schon gelernt, dass Stringlisten recht einfach mit **LoadFromFile** und **SaveToFile** gespeichert werden können. Wenn man allerdings eigene Datensätze definiert, muss man sich selbst um das Speichern und Laden kümmern. Das geschieht in den meisten Programmiersprachen (und so auch in Pascal) in mehreren getrennten Schritten:

- Definition einer Dateivariablen
- Verbinden der Dateivariablen mit einem Dateinamen
- Öffnen der Datei
- Schreiben/Lesen in die Datei
- Schließen der Datei

Einige der Schritte sind bei unterschiedlichen Dateitypen identisch, andere unterschiedlich. Aus Platzgründen behandeln wir nur die typisierten Dateien mit diesen können sie Records speichern. Es gibt noch zwei weitere Dateitypen in Pascal die untypisierten Dateien (bei denen sie auf jedes Byte einer Datei einzeln zugreifen können) und Textdateien (wenn sie einen Text einlesen). Weiterhin gibt es noch weitere auf diesen Basistypen aufbauende höhere Dateiformate. Aus Zeitgründen können diese nicht besprochen werden.

Schritt 1: Dateivariablen deklarieren

Das Erste ist es, in der Variablendefinition eine Dateivariablen zu deklarieren. Eine Dateivariablen ist eine Variablen über die der Zugriff auf eine Datei möglich ist. Sie ist das Bindeglied zwischen den Daten auf der Festplatte und den Daten im Arbeitsspeicher.

Es gibt Dateivariablen für unterschiedliche Daten Für Rekords werden nur **typisierte Dateien** verwendet, das sind Dateien mit einer festgelegten Struktur, die abhängig vom Datentyp ist. In diese Dateien können nur Variablen mit diesem Datentyp gespeichert werden. Die Deklaration geht folgendermaßen:

Var <dateivariablen> : File of <Datentyp>

Zum Beispiel:

```
var Datei : file of kontorecord;
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

Dies deklariert eine Datei, in die unsere Kontodaten gespeichert werden können. (Sie nimmt Kontorecords auf, keine anderen Daten!)

Dies geht auch mit anderen Datentypen. So könnten sie definieren:

```
type Integerdatei : File of Integer;
```

um Integerzahlen zu speichern (in binärer Form, nicht als lesbare Darstellung).

Schritt 2: Datei mit Dateinamen verknüpfen

Das Zweite ist es die Datei, die nur eine Struktur im Speicher ist, mit dem Dateisystem auf der Festplatte zu verknüpfen. Dies ist bei allen Dateien gleich und geschieht mit der **AssignFile** Anweisung:

```
AssignFile(<dateivariabel>,<Dateiname>);
```

also z.B.:

```
AssignFile(datei,'C:\Windows\meine Datei.rec');
```

Der Dateiname ist eine normale Stringvariable, die natürlich auch nur Zeichen enthält, die in Dateinamen gültig sind. (so sind ?,* und : zum Beispiel nicht möglich). Eine Pfadangabe ist möglich. Fehlt sie, so wird das letzte von der Anwendung benutzte Verzeichnis verwendet. Die Dateivariabel ist die Variable, die vorher in der VAR-Sektion deklariert wurde.

Wichtig: Sie müssen das Recht haben die Datei anzulegen. In den Übungen bei der Dualen Hochschule sollten sie z.B. ihr H: Laufwerk benutzen. Wenn sie einen Pfad zur Datei angeben, so muss der Pfad existieren.

Schritt 3: Datei öffnen

Das Öffnen von Dateien erfolgt nun wieder unterschiedlich bei den verschiedenen Dateitypen. Bei typisierten Dateien können die Prozeduren **Rewrite(<dateivariabel>)** und **Reset(<dateivariabel>)** verwendet werden:

- Rewrite erzeugt eine Neue Datei. Diese ist danach leer. Gab es vorher eine Datei mit diesem Namen und einem Inhalt, so ist der Inhalt danach gelöscht. Sie müssen das Schreibrecht auf diese Datei haben, wenn sie schon existiert. Bei Rewrite muss die Datei aber nicht vorher vorhanden sein, dann wird eine neue Datei erzeugt.
- Reset öffnet eine Datei. **Diese muss vorher existieren.** In diese Datei kann danach geschrieben werden oder aus ihr gelesen werden. Es gibt einen Fehler, wenn die Datei nicht existiert.

Erst nach dem Erzeugen der Datei auf diese Weise kann auf die Datei zugegriffen werden!

Schritt 4: In Dateien lesen und schreiben

Das Besondere an typisierten Dateien ist der wahlfreie Zugriff. Darunter versteht man, dass man einen beliebigen Satz lesen kann, ohne alle Sätze vorher zu kennen.

Anhand des bei der Deklaration angegebenen Datentyps kann das System errechnen, wo z.B. der 100.ste Satz auf der Festplatte liegt. Das geht nur, wenn jeder Datensatz gleich groß ist. **Daher muss der Datentyp angegeben werden und der Datentyp darf keine variable Länge haben. Enthält er z.B. Strings, so muss bei der Definition deren maximale Länge angegeben werden.**

Dies ist bei der Stringdefinition möglich. Ändert man die Definition wie folgt ab, so hat der Kontorecord eine feste Länge.

```
type
  Kontorecord = record
    Kontonummer : integer;
    kontostand  : double;
    KontoInhaber: string[20];
end;
```

Dabei ist der KontoInhaber ein String der maximal 20 Zeichen lang ist (er kann auch kürzer sein, jedoch nie länger). Dadurch kann der Compiler errechnen, dass dieser Datensatz genau 32 Bytes groß ist (20 Bytes für den String, 8 für die Double und 4 für die Integer Variable). Pascal verwaltet intern einen Zeiger, der die Position in der Datei angibt. Anfangs steht dieser auf 0, d.h. am Dateianfang. Es gibt nun mehrere Möglichkeiten Daten zu lesen:

Read und Write: Dies geht wie von der Konsole, nur eben aus der Datei. Dabei muss die Dateivariablen und eine Variable des Satztyps angegeben werden:

Read(<dateivariablen>,<variablen>);

Write(<dateivariablen>,<variablen>);

Es kann auch mehr als eine Variable angegeben werden. Dann wird jeder Satz in eine Variable eingelesen. Readln geht nicht, da diese Prozedur nur für Textdateien ausgelegt ist. Lässt man die Dateivariablen weg, so liest und schreibt man auf den Bildschirm.

```
program Project3;
{$APPTYPE CONSOLE}

uses
  SysUtils;

type
  Kontorecord = record
    Kontonummer : integer;
    kontostand  : double;
    KontoInhaber: string[20];
end;

var
  konto: Kontorecord;
  Datei: file of Kontorecord; // Dateivariablen deklarieren

begin
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
AssignFile(datei, 'C:\Windows\meine Datei.rec');
// Datei mit Dateinamen verbinden
Reset(datei); // Datei öffnen
while not eof(datei) do // Lesen bis zum Dateiende
begin
  Read(datei, konto); // Satz lesen
  writeln('Konto Nr. ', konto.Kontonummer); // auf Konsole ausgeben
  writeln('Kontostand', konto.Kontostand);
  writeln('Inhaber . ', konto.Kontoinhaber);
end;
CloseFile(datei); // Datei schließen
end.
```

Die Read Anweisung nimmt eine Dateivariablen und eine Variable, in der ein Datensatz landet, entgegen. Sie liest einen Satz ein und schiebt den Zeiger auf den nächsten Satz, sodass die Datei sequentiell gelesen wird.

Die Funktion **eof()** nimmt eine Dateivariablen als Parameter entgegen und prüft, ob der Zeiger schon am Dateiende ist. Wenn ja so gibt sie True zurück (eof = End of File), wenn nicht so gibt sie false zurück. Daher muss man die Bedingung für eine while Schleife mit Not negieren.

Das Schreiben gestaltet sich einfacher. Da man weiß, wie viele Records geschrieben werden sollen. Im folgenden Beispiel soll in der Variable Anzahl die Nummer der belegten Datensätze aufnehmen.

```
type MyRecordType = record
  TheName : string [40];
  TheIncome: currency;
  ThId : Cardinal;
end;

var thedata : array [1..1000] of MyRecordType;
    anzahl : cardinal;

Procedure writeFile(const filename : string);

var myfile : file of MyRecordType;
    i : Cardinal;

begin
  AssignFile(myfile, filename); // Zuordnung von Datei und Dateiname - als Parameter
  übergeben
  Rewrite(myfile); // Öffnen der Datei zum Schreiben - Rewrite anstatt Reset
  beim Lesen
  For i:=1 to Anzahl do
    write(myfile, thedata[i]); // Schreiben aller Records in einer Schleife
  CloseFile(myfile); // Datei schließen
end;
```

Dies sind auch die wichtigsten Funktionen zum Lesen und Schreiben. Will man auf einen beliebigen Satz zugreifen so geht dies mit folgenden Funktionen:

- Seek(<dateivariablen>, <zahl>) : Setzt den Satzzeiger auf diese Zahl damit kann die Lese/Schreibposition beliebig innerhalb der Datei verschoben werden. Der erste Satz hat die Position 0. Beim Lesen mit Read oder Write wandert sie jeweils um 1 nach hinten.

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

- **Filepos(<dateivariable>)** : integer: Liefert dagegen die aktuelle Position des Satzzeigers zurück. **Seek(datei,filepos(datei)+1)** würde ihn z.B um einen Satz verschieben.
- **Filesize(<dateivariable>)** : integer: Liefert die Größe der Datei in Sätzen zurück. Eine leere Datei hat 0 Sätze Länge. Mit **Seek(Datei,Filesize(Datei))** setzen sie den Lesezeiger an das Dateiende und können bei einer mit Reset geöffneten Datei neue Sätze anfügen.

Schritt 5: Dateien schließen

Das Letzte ist nun die offene Datei zu schließen. Um die Geschwindigkeit zu steigern, führte Delphi alle Operationen zuerst im Arbeitsspeicher durch. Wenn die Datei geschlossen wird, so werden alle Änderungen auf die Festplatte geschrieben. Danach kann man nicht mehr mit read oder write auf die Datei zugreifen sondern muss sie erneut öffnen. Das Schließen führt man mit **CloseFile(<dateivariable>);** durch.

Das Schreiben auf die Festplatte ohne die Datei zu schließen (z.B. bei Programmen, die dauernd laufen und nicht Daten durch Programmabstürze, Neuboots etc. verlieren dürfen kann man durch die Prozedur **Flush(<dateivariable>);** erzwingen.

Dateien verwalten

Um Dateien zu löschen, umzubenennen etc. braucht man sie nicht öffnen. Dafür gibt es einige einfache Funktionen:

- **FileExists(<Dateiname>)** : **Boolean** - gibt True zurück, wenn die Datei existiert und false wenn nicht. Der Dateiname ist (wie bei allen folgenden Beispielen) ein String.
- **Renamefile(<alter name>,<neuer name>)**: Boolean benennt die Datei um. Ist dies möglich, so wird true zurückgegeben. False, wenn es einen Fehler gibt (Datei ist offen, existiert nicht).
- **Deletefile(<dateiname>)** : **Boolean** – löscht die entsprechende Datei von der Festplatte.
- **Removedir(<dir>)** : **Boolean** – löscht das entsprechende Verzeichnis von der Festplatte. Es muss vorher leer sein.
- **MkDir(<dir>)** : **Boolean** – erzeugt das entsprechende Verzeichnis.
- **ChDir(<dir>)** : Wechselt in dieses Verzeichnis, Dateien mit Dateinamen ohne explizite Pfadangabe werden nun dort gespeichert.

Mit den Funktionen Extractfilepath, Extractfilename und Extractfileext kann man von einem Dateinamen den Pfad, den Namen und die Erweiterung extrahieren:

Extractfilepath('C:\Verzeichnis\Datei.txt') ergibt 'C:\Verzeichnis'

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

`Extractfilename('C:\Verzeichnis\Datei.txt')` ergibt 'Datei.txt'

`Extractfileext('C:\Verzeichnis\Datei.txt')` ergibt '.txt'

Das Dateihandling ist durch die verschiedenen Schritte recht kompliziert. Dafür gibt es in zur Vertiefung ein Demoprojekt.

Textdateien

Die nun folgenden Typen von Dateien werden in der Vorlesung und Übung nicht benötigt, sind jedoch vielleicht für eigene Projekte wichtig.

Textdateien speichern die Daten in lesbarer Form. Bei den typisierten Dateien werden die Daten so abgelegt, wie sie im Speicher vorliegender Double Variable sind es z.B. immer acht Bytes egal, ob der Wert „15“ beträgt oder „15.37634764“.

Die grundsätzliche Vorgehensweise bei Textdateien ist identisch zu den typisierten Dateien. Auch hier müssen Dateien mit einer Dateivariablen verknüpft, geöffnet und geschlossen werden. Es gibt aber Unterschiede:

- Der Dateitype ist immer „Textfile“. Dieser Typ steht für eine Textdatei
- Es steht neben den Operationen Reset und Rewrite auch noch die Operation **Append** zur Verfügung: Sie öffnet eine Textdatei und fügt neue Daten am Ende ein. Auch bei Reset gibt es einen Unterschied: Anders als bei den typisierten Dateien kann mit Reset nur gelesen werden. Geschrieben wird mit Rewrite und Append.
- Das Schreiben geht mit den Operationen Write und Writeln, das Lesen mit Read und Readln

Im Prinzip funktioniert das Schreiben in eine Datei genauso wie auf den Bildschirm. Dies ist von den Programmen im ersten Semester noch bekannt. Es ist auch möglich, die Doppelpunkte zur Formatierung zu benutzen um die Vor- und Nachkommastellen einer Zahl festzulegen.

Es ist möglich, mehrere Werte in einer Zeile mit Write zu schreiben und mit Read zu lesen. Sie werden dann durch Kommas getrennt. Das geht, wenn alle Werte Zahlentypen sind. Bei Zahlen kann der Parser das Ende feststellen, wenn er auf Leerzeichen trifft, die durch das Komma in die Ausgabe eingefügt werden. Bei Strings ist dies nicht möglich. Hier wird eine komplette Zeile eingelesen. Daher sollten Strings in eine eigene Zeile geschrieben werden.

Bei Strings ist daher das Schreiben und Lesen mit Readln und Writeln nötig. Es ist aber auch bei Zahlen anzuraten, da so der Inhalt der Datei besser für Menschen lesbar ist (z.B. bei der Fehlersuche).

Das folgende Beispiel zeigt ein minimales Programm. Bei Typen die keine Zahlen oder Strings sind (Aufzählungstypen, Boolean) muss mit **Ord()** eine Konvertierung erfolgen, bevor man schreibt und eine Zahl eingelesen und mit **Typenname()** die Rückkonvertierung erfolgen. (Typenname()) entspricht dem vereinbarten Datentyp, z.B. Boolean().

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
program demotextdateien;  
{ $APPTYPE CONSOLE }  
  
uses  
  SysUtils;  
  
var  
  a : integer;  
  b : double;  
  c : String;  
  d : char;  
  e : boolean;  
  
  datei : textfile;  
  
begin  
  a:=47; b:=pi;  
  c:='Hallo Welt';  
  d:='#';  
  e:=true;  
  Assignfile(datei, 'demodatei.txt');  
  Rewrite(datei);  
  writeln(datei, a);  
  writeln(datei, b);  
  writeln(datei, c);  
  writeln(datei, d);  
  writeln(datei, ord(e));  
  Closefile(datei);  
end.
```

Der Inhalt der Datei:

47

3.14159265358979E+0000

Hallo Welt

#

1

Untypisierte Dateien

Mit diesem Dateityp werden binäre Daten eingelesen oder unbekannte Daten eingelesen und in eigene Daten umgewandelt (z.B. die Rohdaten eines Bildes oder der Spielstand eines Spiels).

Auch hier ist die Vorgehensweise grundsätzlich gleich zu typisierten Dateien, es gibt aber kleine Unterschiede:

- Der Dateityp ist File. Es wird nicht näher spezifiziert, welche Daten die Datei enthält
- Sie können mit Reset und Rewrite lesend und schreibend zugreifen, ebenso funktionieren Seek und Filesize wie bei den typisierten Dateien.
- Wie groß ein Datensatz ist, geben sie beim Öffnen mit einer Zahl an, wird sie weggelassen so entspricht ein Datensatz 128 Byte. In der Regel liest man Dateien byteweise, oder jeweils 16 oder 32 Bit, abhängig von der Struktur der Daten. So würde man eine Datei mit den Instruktionen Reset(datei,1), Reset(datei,2) oder Reset(datei,4) öffnen.
- Das Lesen und Schreiben erfolgt mit zwei neuen Prozeduren: Blockread und Blockwrite.

Blockwrite hat folgende Syntax:

Blockwrite(<dateivariablen>,<variablen>,<Anzahl an Datensätzen>);

Die Instruktion schreibt in die Datei nicht nur einen, sondern so viele Datensätze, wie vorgegeben. So kann man mit einer Instruktion z.B. ein ganzes Array schreiben (siehe Beispiel 1). Mittels der Funktion Sizeof() kann man bei jeder Variable ermitteln, wie viele Bytes sie im Speicher belegt.

Blockread ist fast identisch, hat jedoch noch einen weiteren Parameter:

Blockread(<dateivariablen>,<variablen>,<Anzahl an Datensätzen>,<gelesene Datensätze>);

In der als vierter Parameter übergebenen Variable, landen die tatsächlich gelesenen Datensätze. Wird das Dateiende erreicht, so ist diese Zahl natürlich kleiner als die Anzahl, die man zu lesen wünscht, ansonsten gleich groß.

Zwei Beispiele zeigen die Benutzung von untypisierten Dateien. Beispiel 1 speichert ein Array von Fließkommazahlen in einer Datei:

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
program demountypisiertedateien;  
{ $APPTYPE CONSOLE }  
  
uses  
  SysUtils;  
  
var  
  a : array of double;  
  anzahl : integer;  
  i : cardinal;  
  datei : File;  
  
begin  
  write('wie groß soll das Array sein? '); readln(anzahl);  
  setlength(a,anzahl);  
  for i:=0 to high(a) do  
    a[i]:=random*i;  
  Assignfile(datei,'demodatei.bin');  
  Rewrite(datei,1);  
  Blockwrite(datei,a[0],sizeof(a[0])*length(a));  
  Closefile(datei);  
end.
```

Es gibt zwei Dinge zu beachten: Beim Speichern muss das erste Array Element angegeben werden und nicht die Adresse des Arrays selbst (A[0] und nicht A). Bei dynamischen Arrays ist die Konstruktion `sizeof(a[0])*length(a)` nötig, da `sizeof` nur die Definitionsgröße des Arrays bekommt, nicht die reale Größe. Bei statischen Arrays (Größe ist definiert in der Art Array [1..100], siehe Beispiel 2) muss dagegen die Angabe nur `Sizeof()` lauten, da hier Definitionsgröße und reale Größe gleich groß sind.

Das zweite Beispiel liest diese Datei ein und gibt den Inhalt also sogenannten Hexdump (basierend auf dem Zahlensystem 16, die Buchstaben A-F stehen für die Zahlen 10 bis 15) aus. Rechts stehen die Byte Werte als Buchstabencodes. Da die Codes unter 32 für Steuerzeichen stehen, (nicht druckbare Zeichen) wird für diese ein Punkt ausgegeben.

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
program demountypisiertedateien2;
{$APPTYPE CONSOLE}

uses
  SysUtils;

var a : array [1..16] of byte;
    anzahl : integer;
    datei : File;

Procedure Hexwrite(anzahl : integer);
var i : integer;
    s : string;
begin
  for i:=1 to anzahl do
    begin
      s:=inttohex(a[i],2)+' ';
      write(s);
    end;
    write (' | ');
  for i:=1 to anzahl do
    if a[i]<32 then write('.') else write(char(a[i]));
  writeln;
end;

begin
  Assignfile(datei,'demodatei.bin');
  Reset(datei,1);
  repeat
    Blockread(datei,a[1],sizeof(a),anzahl);
    if anzahl>0 then Hexwrite(anzahl);
  until anzahl<sizeof(a);
  Closefile(datei);
  Readln;
end.
```

Ini Dateien

Eine sehr bequeme Möglichkeit Dateien zu erzeugen, die einfach lesbar und einfach durch ein Programm verarbeitbar sind, sind Ini Dateien. Sie dienen dazu, Einstellungen zu speichern, können aber auch genutzt werden, um eigene Daten zu speichern, die nicht strukturiert sind.

Der allgemeine Code für die Verwendung einer Inidatei sieht so aus:

- Erzeugen eines Inidatei Objektes mit `Create(dateiname)`
- Lesen/Schreiben von Werten
- Freigeben des Inidatei Objektes mit `Free`

Am Beispielcode auf der nächsten Seite wird dies leicht ersichtlich. In die Usesliste muss die Unit „inifiles“ aufgenommen werden. Es muss ein Dateiobjekt vom Typ `Tinifile` existieren (hier: „inidatei“) Eine Datei wird dann erzeugt mit `Datenobjekt:=Tinifile.create(Filename);` wobei `Filename` der Dateiname ist. Das Ergebnis wird dann dem Dateiobjekt zugewiesen.

Dieses Dateiobjekt verfügt dann über die Methoden `WriteString`, `WriteFloat`, `WriteInteger` und `WriteBool`, welche diese Werte schreiben können und entsprechende Lesemethoden mit `ReadXXXX`.

Der Erste Parameter ist der Sektionsname. An einem Dump wird sichtbar, wie die Datei aufgebaut ist:

```
[Einstellungen]
Titel=Mein Formular
Left=100
Datum=40314,5536212963
```

„Einstellungen“ ist der Sektionsname. Das erlaubt es, gleichlautende Schlüssel in mehreren Sektionen zu haben oder die Datei weiter zu strukturieren.

Es folgt dann der Schlüsselname, das ist der Name vor dem „=“ in der Datei. Hier also „Titel“, „Left“ und „Datum“. Hinter dem Schlüsselnamen folgt dann der Wert von Variablen. Wie am Dump erkennbar, können auch Menschen die Daten lesen und gegebenenfalls verändern. Weiterhin muss nicht die ganze Datei ausgelesen und geschrieben werden sondern nur die Schlüssel, die verändert wurden.

Die Lesemethoden unterscheiden sich von den Schreibmethoden durch einen vierten Parameter, dem Vorgabewert: Fehlt der Schlüsselname in der Datei, so wird der Vorgabewert zurückgegeben. Inidateien können nur in bestimmten Verzeichnissen geschrieben werden, nämlich den für Anwendungsdaten vorgesehenen. Hier wurde hier

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

über Anfragen der Umgebungsvariable „Appdata“ erfragt.

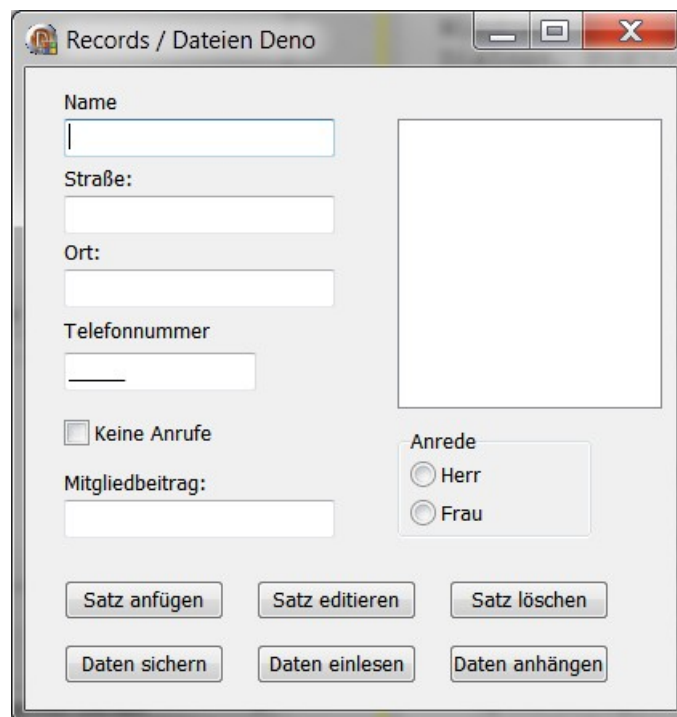
```
unit inidateiendemo;  
  
interface  
  
uses  
  windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,  
  Dialogs, StdCtrls;  
  
type  
  TForm4 = class(TForm)  
    Label1: TLabel;  
    Button1: TButton;  
    procedure FormCreate(Sender: TObject);  
    procedure Button1Click(Sender: TObject);  
  private  
    { Private-Deklarationen }  
  public  
    { Public-Deklarationen }  
  end;  
  
var  
  Form4: TForm4;  
  
implementation  
  {$R *.dfm}  
  
uses inifiles;  
  
procedure TForm4.Button1Click(Sender: TObject);  
var inidatei : Tinifile;  
    anwendungsdaten : string;  
  
begin  
  anwendungsdaten:= GetEnvironmentVariable('Appdata');  
  inidatei:=Tinifile.Create(anwendungsdaten+'\Einstellungen1.ini');  
  try  
    Inidatei.WriteString('Einstellungen', 'Titel', Caption);  
    Inidatei.WriteInteger('Einstellungen', 'Left', left);  
    Inidatei.WriteFloat('Einstellungen', 'Datum', Now);  
  finally  
    Inidatei.Free;  
  end;  
end;  
  
procedure TForm4.FormCreate(Sender: TObject);  
var inidatei : Tinifile;  
    anwendungsdaten : string;  
  
begin  
  anwendungsdaten:= GetEnvironmentVariable('Appdata');  
  inidatei:=Tinifile.Create(anwendungsdaten+'\Einstellungen1.ini');  
  form4.Caption:=Inidatei.ReadString('Einstellungen', 'Titel', '');  
  form4.Left:=inidatei.ReadInteger('Einstellungen', 'Left', 100);  
  label1.Caption:=Datetimetostr(inidatei.ReadFloat('Einstellungen', 'Datum', Now));  
  Inidatei.Free;  
end;  
  
end.
```

Das schreiben einer größeren Anwendung

Um die Verarbeitung von Records und Dateien zu zeigen schreiben wir eine kleine Demoanwendung. Die heutige Anwendung soll stellvertretend für die Verwaltung von Daten stehen.

Damit dies nicht allzulange dauert bauen wir nur ein einfaches Formular, das die wichtigsten Funktionen die ein Sportclub braucht umfasst.

So sieht das Formular aus:



Name, Straße, Ort und Mitgliedsbeitrag sind LabeledEdit Felder, Telefonnummer ein Maskedit. Die anderen Elemente sind eine Checkbox für „Keinen Mitgliedsbeitrag und ein Radiogroup Feld. Damit sieht die Deklaration so aus:

```
TForm4 = class(TForm)
  vorname_name: TLabelledEdit;
  Strasse: TLabelledEdit;
  liste: TListBox;
  Anfüegenbutton: TButton;
  Speichernbutton: TButton;
  Einlesenbutton: TButton;
  OpenDialog1: TOpenDialog;
  SaveDialog1: TSaveDialog;
  erweiternbutton: TButton;
  Telefonnummer: TMaskEdit;
  adresse: TLabelledEdit;
  Label1: TLabel;
  anrede: TRadioGroup;
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
keine_anrufe: TCheckBox;  
mitgliedsbeitrag: TLabelEdit;
```

Einige Prüfroutrinen für die Eingabe sollen implementiert werden.

- Wenn die Checkbox „keine anrufe“ markiert ist, dann ist das Feld für Telefonnummer ausgegraut (enabled=true);
- Es gibt drei mögliche Mitgliedsbeiträge:
Kinder und Jugendliche: 25 Euro
Erwachsene: 50 Euro
Familien: 70 Euro
Andere Beträge sollen nicht eingegeben werden können.
- Eines der beiden Anredefelder muss aktiv sein.

Wo sollten Prüfungen implementiert werden? (welche Ereignisse nutzen sie aus?).
Versuchen sie sich an den Routinen und begründen sie wo sie diese implementiert haben.

In dem Formular gibt es noch eine Sache zu tun: Die Listbox rechts (mit dem Namen „Liste“) nimmt den Namen auf. Später soll man durch Anklicken auf einen Namen den entsprechenden Eintrag auswählen können und die Anzeige wird aktualisiert.

Fügen sie den Code hinzu, der die Liste beim Klicken auf „Satz anfügen“ erweitert.

Intern speichern wir alle Daten der linken Seite in einem Record mit folgender Definition:

```
Type Mitgliedsrecord = record  
  Mitglied : String [30];  
  Strasse : String [25];  
  Telefon: Integer;  
  ort: String [25];  
  Beitrag : Currency;  
  Anrede: Integer;  
  Keine_Anrufe: Boolean;  
end;
```

Dies ist eine 1:1 Umsetzung der relevanten Datentypen der Eingaben bzw. entspricht der Logik (beim Mitgliedsbeitrag). Es sind sinnvoll zwei Routinen zu schreiben mit folgenden Aufgaben:

Procedure Kopiere_in_Record

legt die Daten die im Formular sind in einer lokalen Variable vom Typ Mitgliedsrecord ab.
Und

Procedure Kopiere_in_Formular

macht das umgekehrte. Dies erlaubt es uns zum einen Ansicht von interner Datenhaltung zu trennen, und zum anderen müssen wir nun nur noch mit einer Variablen arbeiten anstatt den Eingaben von 7 Feldern. Diese Prozeduren werden zusätzlich zu den normalen deklariert in der public Sektion. So sieht dann der Code aus:

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
public
  mitglied : Mitgliedsrecord;
  Procedure Kopiere_in_Record;
  Procedure Kopiere_in_Formular;
  { Public-Deklarationen }
end;

var
  Form4: TForm4;

implementation

{$R *.dfm}
Procedure TForm4.Kopiere_in_Record;

begin
  mitglied.Mitglied:=vorname_name.Text;
  mitglied.Strasse:=Strasse.Text;
  mitglied.Telefon:=Strtoint(Telefonnummer.Text);
  mitglied.Ort:=adresse.Text;
  mitglied.Beitrag:=strtofloat(mitgliedsbeitrag.Text);
  mitglied.Anrede:=anrede.ItemIndex;
  mitglied.Keine_Anrufe:=keine_anrufe.Checked;
end;

Procedure TForm4.Kopiere_in_Formular;

begin
  vorname_name.Text:=mitglied.Mitglied;
  Strasse.Text:=mitglied.Strasse;
  Telefonnummer.Text:=inttostr(mitglied.Telefon);
  adresse.Text:=mitglied.Ort;
  mitgliedsbeitrag.Text:=floattostr(mitglied.Beitrag);
  anrede.ItemIndex:=mitglied.Anrede;
  keine_anrufe.Checked:=mitglied.Keine_Anrufe;
end;
```

Die erste Routine müssen wir aufrufen, wenn wir auf den Button „Satz anfügen“ klicken. Die Bedingung für das Aktualisieren ist dagegen nicht sofort sichtbar. Aber klar ist, dass die Ausgabe aktualisiert werden muss, wenn wir einen Eintrag in der Liste rechts klicken. Zusammen mit den Prüfbedingungen sieht nun der Code für den Anfügenbutton so aus:

```
procedure TForm4.AnfuegenbuttonClick(Sender: TObject);

var
  beitrag : currency;
  fehler : string;

begin
  beitrag:=StrToFloat(mitgliedsbeitrag.Text);
  if not((beitrag=25) or (beitrag=50) or (beitrag=70)) then fehler:='Fehlerhafter
Beitrag'#13 else fehler:='';
  if anrede.ItemIndex<0 then fehler:=fehler+'Anrede muss markiert sein'#13;
  if fehler<>' ' then
  begin
    Showmessage(fehler);
    exit;
  end;
  Kopiere_in_record;
  liste.Items.Add(vorname_name.Text);
  setlength(mitglieder, length(mitglieder)+1); // Platz für neuen Rekord schaffen
  mitglieder[high(mitglieder)]:=mitglied;
  liste.ItemIndex:=liste.Count-1;
end;
```

und beim Onclick Ereignis für die Liste:

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
procedure TForm4.listeClick(Sender: TObject);  
begin  
    Kopiere_in_Formular;  
end;
```

Der Sprung von einem Record auf viele ist nun recht einfach. Es muss nur ein Array deklariert werden. Da die Anzahl nicht feststeht ist ein dynamisches Array sinnvoll. Welcher Eintrag gerade aktiv ist, kann man über Itemindex auslesen, der Index geht von 0 los, wie bei dynamischen Arrays, das erleichtert die Sache ganz enorm. Beim Hinzufügen eines Satzes muss dann nur das Array vergrößert werden und der Satz im Arrays gespeichert.:

```
setlength(mitglieder,length(mitglieder)+1); // Platz für neuen Rekord schaffen  
mitglieder[high(mitglieder)]:=mitglied; // Record ins Array kopieren
```

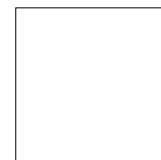
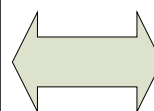
Diese beiden Zeilen fügen sie in die Anfügen Button OnclickRoutine ein:

und die ListeOnClick Routine sieht nun so aus:

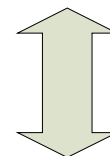
```
procedure TForm4.listeClick(Sender: TObject);  
begin  
    mitglied:=mitglieder[liste.ItemIndex];  
    Kopiere_in_Formular;  
end;
```

Grafisch sieht die Vorgehensweise nun so aus:

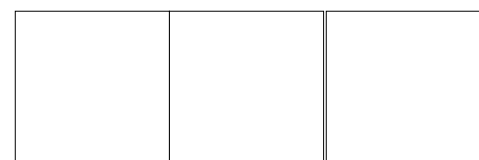
Laden1Click /
Speichern1Click



Rekord



Adr[...]:=record /
record:=Adr[..]



Array Element 1 Array Element 2 Array Element 3

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

Wir tauschen mit dem Array über den Zwischenrekord Daten aus. Er nimmt die momentan sichtbaren Daten auf. Damit haben wir Oberfläche von Datenhaltung entkoppelt, was die Pflege der Anwendungen einfacher macht. Das Dateihandling orientiert sich nun nach den Routinen die wir bei Dateien durchgenommen haben. So werden die Daten gespeichert (Routine EinlesenbuttonClick):

```
procedure TForm4.EinlesenbuttonClick(Sender: TObject);
var datei : file of Mitgliedsrecord;

begin
  if opendialog1.Execute then
  begin
    assignfile(datei, opendialog1.FileName); // Datei mit dem gewählten Dateinamen
    verknüpfen
    Reset(datei); // Datei zum Lesen öffnen
    setlength(mitglieder, 0); // Array löschen
    liste.items.clear; // Listbox löschen
    while not eof(datei) do
    begin
      setlength(mitglieder, length(mitglieder)+1); // Platz für neuen Rekord schaffen
      read(datei, mitglieder[high(mitglieder)]); // Satz ans Arrayende anfügen
      liste.items.Add(mitglieder[high(mitglieder)].Mitglied); // Satz in die Listbox
    einfügen
    end;
    Closefile(datei);
  end;
end;
```

Neu ist die Verwendung eines Opendialogs. Er wird zum Formular hinzugenommen und erst im Programm aufgerufen:

```
if opendialog1.Execute then
```

Der Rückgabewert der Funktion ist True, wenn der Anwender den Dialog mit „ok“ schließt und false, wenn er auf „abbruch“ klickt. Also speichern wir nur wenn der Rückgabewert true ist.

Man kann einige Dinge einstellen:

title: Beschriftung des Dialogs

initaldir: Verzeichnis das beim Start geöffnet wird

filter: Eine Liste von Einschränkungen der anzeigbaren Dateinamen. Ein Pipe Symbol trennt Beschreibung und Filter sowie verschiedene Einträge:

```
opendialog1.Filter:='Datendateien|*.dat|Alle Dateien|*.*';
```

definiert zwei Einträge. Die für den Anwender lesbare Beschriftung sind „Datendateien“ und „Alle Dateien“. Der Filter, welcher Einträge ausblendet ist „.dat“ und „*.*“.

Filterindex: Wählt den Eintrag aus der Liste aus, der beim Start aktiv ist.

Analog funktioniert auch ein Speicherdialog, der jedoch z.B. Nachfragen stellt wenn man eine schon existierende Datei wählt. Analog entsteht dann die Speichernroutine:

```
procedure TForm4.SpeichernbuttonClick(Sender: TObject);
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
var datei : file of Mitgliedsrecord;
    i : integer;

begin
  if savedialog1.Execute then
  begin
    assignfile(datei,savedialog1.FileName); // Datei mit dem gewählten Dateinamen
    verknüpfen
    Rewrite(datei); // Datei zum Schreiben öffnen
    for i:=low(mitglieder) to high(mitglieder) do
    begin
      write(datei,mitglieder[i]); // Sätze sequentiell schreiben
    end;
    Closefile(datei);
  end;
end;
```

Von ihr abgeleitet ist die Anfügenroutine, die Daten von der Festplatte um die schon existierenden in der Anwendung ergänzt. Sie unterscheidet sich eigentlich nur durch den Startindex, der mit seek auf das Dateiende gesetzt wird. Weiterhin wird mit Reset die Datei geöffnet. Bei Rewrite würde der Inhalt gelöscht werden

```
procedure TForm4.erweiternbuttonClick(Sender: TObject);
var datei : file of Mitgliedsrecord;
    i : integer;

begin
  if savedialog1.Execute then
  if fileexists(savedialog1.FileName) then // Datei muss zum Anfügen existieren
  begin
    assignfile(datei,savedialog1.FileName); // Datei mit dem gewählten Dateinamen
    verknüpfen
    Reset(datei); // Datei öffnen - nicht mit Rewrite, da
  sonst der alte Inhalt futsch ist
    seek(datei,filesize(datei)); // Lesezeiger an das Dateiende setzen.
    for i:=low(mitglieder) to high(mitglieder) do
    begin
      write(datei,mitglieder[i]); // Sätze sequentiell schreiben
    end;
    Closefile(datei);
  end;
end;
```

Beim Einlesen gibt es noch einen Schönheitsfehler: die Anzeige wird nicht aktualisiert. Diesen kann man leicht beheben, indem man den Itemindex der Liste auf den ersten Eintrag setzt. Danach wird die OnClickroutine aufgerufen.

```
liste.ItemIndex:=0;
listeClick(sender);
```

Nun gibt es noch die wichtigen Funktionen des Editieren eines Eintrags und des Löschens. Das Editieren ist dabei recht einfach:

- Wir müssen die Daten aus dem Formular in den record kopieren
- und den Record dann an die richtige Position des Arrays ablegen
- und dann noch den Eintrag in der Liste aktualisieren:

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

Fürs erste gibt es eine Routine und die richtige Position erhalten wir indem wir `Liste.ItemIndex` nutzen, denn hier ist ja der aktive Satz markiert. Also ist die Editerroutine folgende:

```
procedure TForm4.EditbuttonClick(Sender: TObject);
begin
  Kopiere_in_record;
  mitglieder[Liste.ItemIndex]:=mitglied;
  liste.Items[Liste.ItemIndex]:=vorname_name.Text;
end;
```

Etwas komplizierter ist das Löschen. Der einfachste Weg ist es, den Eintrag im Array, der gerade markiert ist durch den nächsten zu ersetzen und dies zu wiederholen, bis man am Ende des Arrays angekommen ist. Dann kann man die Länge des Arrays um 1 verkürzen. Zuletzt muss man den Eintrag in der Liste aktualisieren und dabei aufpassen, wenn man den letzten Eintrag löscht, dass dann `ItemIndex` auch korrigiert wird:

```
procedure TForm4.loeschenbuttonClick(Sender: TObject);
var i : integer;
begin
  for I:=liste.ItemIndex to high(mitglieder)-1 do mitglieder[i]:=mitglieder[i+1];
  setlength(mitglieder,length(mitglieder)-1);
  liste.Items.Delete(liste.ItemIndex);
  liste.ItemIndex:=min(liste.Items.Count-1,liste.ItemIndex);
  listeClick(sender);
end;
```

Zuletzt gibt es noch einige Abschlussarbeiten zu erledigen. Das Array sollte beim Programmstart auf 0 gesetzt werden:

```
procedure TForm4.FormCreate(Sender: TObject);
begin
  setlength(mitglieder,0); // Anzahl der mitglieder auf 0 setzen
end;
```

Sie finden das Programm unter:

<http://www.delphi-vorlesung.de/files/inf2/DateienDemoformular.zip>

zum runterladen. Hier der komplette Quelltext:

```
unit DateienDemoformular;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, StdCtrls, ExtCtrls, Mask, math;

Type Mitgliedsrecord = record
  Mitglied : String [30];
  Strasse : String [25];
  Telefon: Integer;
  Ort: String [25];
  Beitrag : Currency;
  Anrede: Integer;
  Keine_Anrufe: Boolean;
end;

type
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
TForm4 = class(TForm)
  vorname_name: TLabelledEdit;
  Strasse: TLabelledEdit;
  liste: TListBox;
  Anfuegenbutton: TButton;
  Editbutton: TButton;
  Loeschenbutton: TButton;
  OpenFileDialog1: TOpenDialog;
  SaveDialog1: TSaveDialog;
  Telefonnummer: TMaskEdit;
  adresse: TLabelledEdit;
  Label1: TLabel;
  anrede: TRadioGroup;
  Keine_anrufe: TCheckBox;
  mitgliedsbeitrag: TLabelledEdit;
  Button2: TButton;
  Speichernbutton: TButton;
  Einlesenbutton: TButton;
  procedure FormCreate(Sender: TObject);
  procedure AnfuegenbuttonClick(Sender: TObject);
  procedure listeClick(Sender: TObject);
  procedure EinlesenClick(Sender: TObject);
  procedure SpeichernClick(Sender: TObject);
  procedure erweiterbuttonClick(Sender: TObject);
  procedure Keine_anrufeClick(Sender: TObject);
  procedure EditbuttonClick(Sender: TObject);
  procedure LoeschenbuttonClick(Sender: TObject);
private
  { Private-Deklarationen }
public
  mitglied : Mitgliedsrecord;
  mitglieder : array of Mitgliedsrecord; // dynamisches Array da die Anzahl vorher nicht feststeht
  procedure Kopiere_in_Record;
  procedure Kopiere_in_Formular;
  { Public-Deklarationen }
end;

var
  Form4: TForm4;

implementation

{$R *.dfm}
procedure TForm4.Kopiere_in_Record;
begin
  mitglied.Mitglied:=vorname_name.Text;
  mitglied.Strasse:=Strasse.Text;
  mitglied.Telefon:=Strtoint(Telefonnummer.Text);
  mitglied.Ort:=adresse.Text;
  mitglied.Beitrag:=strtofloat(mitgliedsbeitrag.Text);
  mitglied.Anrede:=anrede.ItemIndex;
  mitglied.Keine_Anrufe:=keine_anrufe.Checked;
end;

procedure TForm4.Keine_anrufeClick(Sender: TObject);
begin
  Telefonnummer.Enabled:=not keine_anrufe.Checked;
end;

procedure TForm4.Kopiere_in_Formular;
begin
  vorname_name.Text:=mitglied.Mitglied;
  Strasse.Text:=mitglied.Strasse;
  Telefonnummer.Text:=inttostr(mitglied.Telefon);
  adresse.Text:=mitglied.Ort;
  mitgliedsbeitrag.Text:=floattostr(mitglied.Beitrag);
  anrede.ItemIndex:=mitglied.Anrede;
  keine_anrufe.Checked:=mitglied.Keine_Anrufe;
end;

procedure TForm4.AnfuegenbuttonClick(Sender: TObject);

var beitrags : currency;
    fehler : string;
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
begin
beitrag:=StrToFloat(mitgliedsbeitrag.Text);
if not((beitrag=25) or (beitrag=50) or (beitrag=70)) then fehler:='Fehlerhafter Beitrag'#13
else fehler:='';
if anrede.ItemIndex<0 then fehler:=fehler+'Anrede muss markiert sein'#13;
if fehler<>'' then
begin
  Showmessage(fehler);
  exit;
end;
Kopiere_in_record;
liste.Items.Add(vorname_name.Text);
setlength(mitglieder,length(mitglieder)+1); // Platz für neuen Rekord schaffen
mitglieder[high(mitglieder)]:=mitglied;
liste.ItemIndex:=liste.Count-1;
end;

procedure TForm4.EditbuttonClick(Sender: TObject);
begin
  Kopiere_in_record;
  mitglieder[list.ItemIndex]:=mitglied;
  liste.Items[list.ItemIndex]:=vorname_name.Text;
end;

procedure TForm4.EinlesenClick(Sender: TObject);
var datei : file of Mitgliedsrecord;

begin
  opendialog1.Filter:='Datendateien|*.dat|Alle Dateien|*.*';
  if opendialog1.Execute then
  begin
    assignfile(datei,opendialog1.FileName); // Datei mit dem gewählten Dateinamen verknüpfen
    Reset(datei); // Datei zum Lesen öffnen
    setlength(mitglieder,0); // Array löschen
    liste.items.clear; // Listbox löschen
    while not eof(datei) do
    begin
      setlength(mitglieder,length(mitglieder)+1); // Platz für neuen Rekord schaffen
      read(datei,mitglieder[high(mitglieder)]); // satz ans Arrayende anfügen
      liste.items.Add(mitglieder[high(mitglieder)].Mitglied); // Satz in die Listbox einfügen
    end;
    closefile(datei);
  end;
  liste.ItemIndex:=0;
  listeclick(sender);
end;

procedure TForm4.erweiternbuttonClick(Sender: TObject);
var datei : file of Mitgliedsrecord;
    i : integer;

begin
  if savedialog1.Execute then
  if fileexists(savedialog1.FileName) then // Datei muss zum Anfügen existieren
  begin
    assignfile(datei,savedialog1.FileName); // Datei mit dem gewählten Dateinamen verknüpfen
    Reset(datei);
    // Datei öffnen - nicht mit Rewrite, da sonst der alte Inhalt futsch ist
    seek(datei,filesize(datei)); // Lesezeiger an das Dateieinde setzen.
    for i:=low(mitglieder) to high(mitglieder) do
    begin
      write(datei,mitglieder[i]); // Sätze sequentiell schreiben
    end;
    closefile(datei);
  end;
end;

procedure TForm4.FormCreate(Sender: TObject);
begin
  setlength(mitglieder,0); // Anzahl der mitglieder auf 0 setzen
end;

procedure TForm4.listeclick(Sender: TObject);
```

Informatik 2 Programmieren in Delphi: Rekords und Typisierte Dateien.

```
begin
  Mitglied := mitglieder[liste.ItemIndex];
  Kopiere_in_Formular;
end;

procedure TForm4.loeschenbuttonClick(Sender: TObject);
var i : integer;
begin
  for I:=liste.ItemIndex to high(mitglieder)-1 do mitglieder[i]:=mitglieder[i+1];
  setlength(mitglieder,length(mitglieder)-1);
  liste.Items.Delete(liste.ItemIndex);
  liste.ItemIndex:=min(liste.Items.Count-1,liste.ItemIndex);
  listeClick(sender);
end;

procedure TForm4.SpeichernClick(Sender: TObject);
var datei : file of Mitgliedsrecord;
    i : integer;
begin
  if savedialog1.Execute then
  begin
    assignfile(datei,savedialog1.FileName); // Datei mit dem gewählten Dateinamen verknüpfen
    Rewrite(datei); // Datei zum Schreiben öffnen
    for i:=low(mitglieder) to high(mitglieder) do
    begin
      write(datei,mitglieder[i]); // Sätze sequentiell schreiben
    end;
    Closefile(datei);
  end;
end;
end.
```